UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

SUPPORT FRAMEWORK FOR AUTOMATED EXPLORATORY DATA ANALYTICS (ADAM)

REALIZADO POR:

Francisco Javier Antón Martínez

DIRIGIDO POR:

Julio Villena Román

DEPARTAMENTO:

Ingeniería Telemática

Leganés, Madrid, septiembre/octubre 2016.

Abstract

In recent decades the amount of data available has increased significantly and, with them, has appeared the figure of the data scientist. The data scientist combines knowledge in the fields of analytics, machine learning, data mining and has experience in programming. However, it is estimated that about 60 % of a data scientist time is used for data cleaning and modeling. The focus of this Final Degree Project is divided in two stages. The first stage tries to tackle this 60 % by performing a series of operations to the input dataset in order to turn any raw dataset into one we can work with in prediction, as well as collect and display as much information about the dataset as possible. Finally, the second stage is developed with the intention that the user introduces the dataset generated in the first stage and, using several Python libraries such as scikit-learn, several algorithms are tested on the data until the best one is found. Stage two will then return the user said algorithm for futures uses.

Abstract

En las últimas décadas la cantidad de datos disponibles ha aumentado significativamente y, con ellos, ha aparecido la figura del científico de datos. El científico de datos es una figura que combina conocimiento en los campos de la analítica, el aprendizaje máquina, la minería de datos, así como experiencia en programación. Sin embargo, se calcula que un 60 % del tiempo de un científico de datos es utilizado para limpieza y modelado de datos. Este trabajo se divide en dos fases. La primera fase trata de apaliar ese 60 % realizando una serie de operaciones a los datos introducidos con el objetivo de convertir cualquier conjunto de datos dado en un conjunto preparado con el que poder hacer trabajos de predicción, así como recopilar y mostrar al usuario tanta información de su conjunto como sea posible. Por otro lado, la segunda fase está desarrollada con la intención de que el usuario introduzca el conjunto de datos generado en la primera fase y, haciendo uso de varias de las librerías de Python como scikit-learn, se prueben sobre él una serie de algoritmos hasta dar con el mejor, que será devuelto al usuario para futuros trabajos de predicción.

Índice

Lista de Figuras

1 Introduction

1.1 Motivation

Big Data is a concept that refers to the storage of massive amounts of data and the procedures used to find patterns within the data.

The discipline dedicated to Big Data falls within the field of Information and Communications Technology (ICT). This discipline takes responsibility for all activities related to systems handling large data sets. The most common difficulties associated with the management of these data are the collection and storage, search, sharing, analysis and visualization. The trend to manipulate huge amounts of data is due to the need, in many cases, to include this information to create statistical reports and predictive models used in several fields such as business analysis, advertising or disease control and prevention.

The upper limit of processing has grown over the years. Thus, the limits set in 2008 were around the order of petabytes or zettabytes [1]. Scientists often find themselves limited in their analysis due to the large amount of data in certain areas such as meteorology, genomics, complex simulations of physical processes and research related to biological and environmental processes.

These limitations also affect Internet search engines, finance systems and business computing. Data sets grow in volume due to the massive data collection from wireless sensors and mobile devices, the steady growth of application logs, cameras, microphones… Technological capacity per capita worldwide to store data doubles, approximately, every forty months since the eighties. It is estimated that in 2012, 2.5 quintillion bytes were created each day [2].

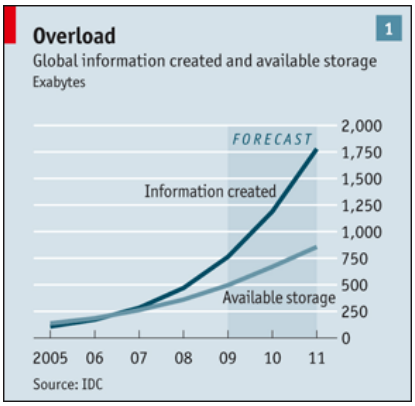

Figure 1.1: Data Overload

In order to deal with all these problems a new field was born: Data Science. Data Science [3] is an interdisciplinary field that involves processes and systems to extract knowledge or a better understanding of large volumes of data in their different forms (both structured and unstructured) and formats.

It is a new paradigm on which researchers rely on systems and processes very different to those used in the past as models, equations, algorithms, as well as evaluation and interpretation of results.

It is estimated that 60 % of the time [4] of a data scientist is spent on cleaning and organizing data. A big part of this work is methodical: variable analysis, test of basic models… For this reason, the creation of an application capable of performing part of this work automatically would be interesting so that data scientists can devote a bigger part of their time to improving their algorithms.

1.2 Objective

The main objective pursued in this Final Degree Project is the creation of a framework capable of automatically analyzing, cleaning and extracting as much information as possible from any given data set. Said framework will be called ADAM (Automated Discovery and Analysis Machine) and it will be divided in two main stages:

- Stage 1 – Data Understanding:
  - Description of the variables contained in the data set.
  - Show main correlation between variables.
  - Variables value support.
  - Creation and analysis of synthetic variables.
- Stage 2 – Data Training:
  - Scoring of predictor variables.
  - Search for the optimal prediction model.

Stage 1 will provide the user with a series of useful statistics regarding his/her data set as well as performing operations of data cleaning for a later usage of the resulting dataframe.

- Basic statistics: A data structure will be created containing as much information of every variable as possible. This information will contain name of the variable, type, amount of missing values, maximum value, minimum value, mean, median, mode, standard deviation and the 25th, 50th, 75th and 100th percentiles. Lastly, a plot of every possible histogram will be displayed.

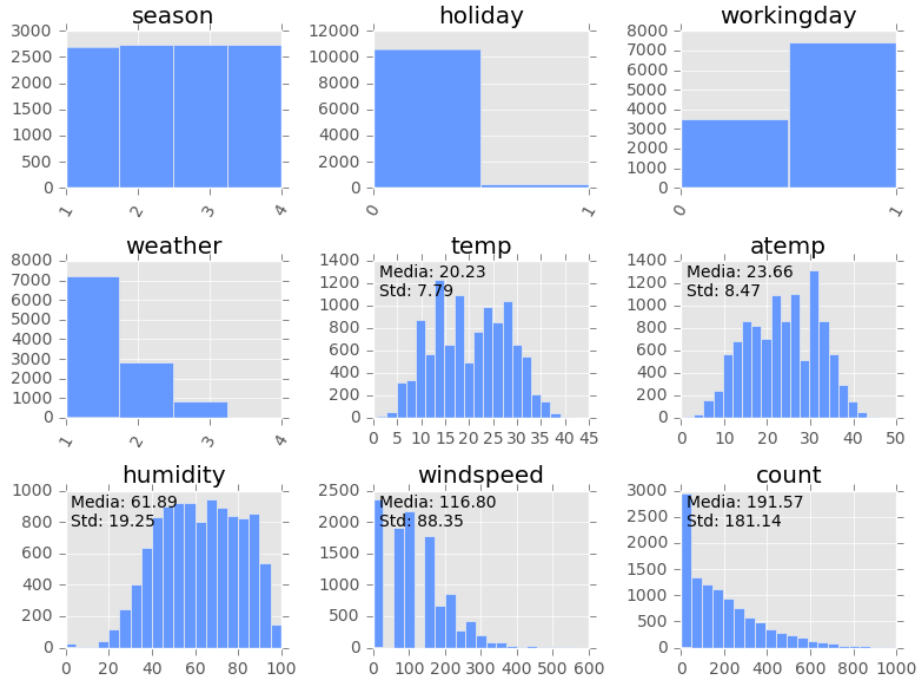

Figure 1.2: Example of a histogram

- Correlations: Main correlations using Pearson's coefficient [5] between variables will be shown. First off, correlations between predictor variables will be displayed. After that two lists of correlations with the target variable will be presented, the first one with the highest correlations (both positive and negative) of every predictor variable with the target variable while the second list will contain every null correlation with the target. Finally, these uncorrelated variables will be dropped since they will not be useful when predicting.

- Variables value support: Computes the support for every possible combination of values. The support is the proportion of items in the data set containing that specific combination.

$$\sup(X) = \frac{|X|}{|D|}$$

- Creation and analysis of synthetic variables: Finally, some synthetic variables will be generated. A synthetic variable [6] is a variable created by performing some kind of operation to two or more of the original variables. This framework will perform operations of addition, subtraction and multiplication of variables and will check whether the correlation of these new variables with the target variable is good enough to keep them or not.

Stage 2 will take a clean dataset and it will return a trained model ready to be used.

- Scoring of predictor variables: A SGD Regressor (Stochastic Gradient Descent) will be performed to the data in order to show the user the variables with the highest predictive value: a ranking of predictive features.

- Search for the optimal prediction model: First off, whether the target variable is categorical or not will be checked in order to perform either regression or classification algorithms. Once that information is known, several algorithms will be tested and the one performing best will be chosen and returned to the user.

It will be made in Python making use of several libraries amongst which it is worth mentioning scikit-learn, a machine learning Python library that allows sophisticated data science work.

It will, also, be tested with some real scenarios to exhibit how it works.

1.3 Legal framework

With the expansion of the Information and Communication Technologies (ICT) and the digital economy, citizens around the globe have lost their ability to control their personal information and maintain their privacy. In January 2012, the European Commission began a process of reform of Community rules regarding personal data protection inside the European Union [7].

ADAM user's must take into account the current legal framework, both Community and country specific when using public data respecting the data protection regulation and the privacy of the citizens at all times.

1.4 Document organization

The rest if this Final Degree Project will be structured in several sections that will be detailed below:

1. The second chapter will study the state of the art (data science).

2. The third chapter will explain the design and implementation of ADAM.

3. The fourth chapter will detail ADAM: Sage 1. It will explain how it works and what to expect out of it.

4. The fifth chapter will describe ADAM: Stage 2. This chapter will be written in a similar manner to the previous to ease the reading.

5. Chapter six will contain several tests done with ADAM.

6. Finally, chapter seven will include the conclusions and discuss some future lines of work.

2 Estado del arte

2.1 Ciencia de datos

Vivimos en un mundo en que se recolectan diariamente grandes cantidades de datos. Analizar esos datos en una necesidad importante.

"Vivimos en la edad de la información" es una frase popular, sin embargo, en realidad vivimos en la edad de los datos. Terabytes o petabytes de datos son arrojados a nuestras redes, la World Wide Web (WWW), y varios aparatos de almacenamiento de datos cada día de trabajo, sociedad, ciencia e ingeniería, medicina, y casi cualquier otro aspecto de nuestra vida diaria.

Este crecimiento explosivo de volumen de datos disponible es el resultado de la informatización de nuestra sociedad y del rápido desarrollo de potentes herramientas de recolección y almacenamiento de datos. Las empresas de todo el mundo generan enormes conjuntos de datos, incluyendo ventas, registros de comercio de acciones, descripciones de productos, promociones de venta, perfiles y rendimiento de empresas y voz del cliente.

Este enorme y ampliamente disponible crecimiento del volumen de datos hace que nuestro tiempo sea en realidad la edad de los datos. Por ello, necesitamos urgentemente de herramientas potentes y versátiles para descubrir automáticamente información valiosa de la tremenda cantidad de datos y transformar dichos datos en conocimiento organizado. Esta necesidad ha dado pie al nacimiento de la minería de datos.

Es un campo joven, dinámico y prometedor. La minería de datos ha empezado y seguirá haciendo grandes avances en nuestro viaje hacia la próxima edad de la información.

En resumen, la abundancia de datos, unida a la necesidad de poderosas herramientas de análisis de datos, ha sido descrita como una situación rica en datos, pero pobre en información. La rápidamente creciente y tremenda cantidad de datos recogida y almacenada en enormes y numerosos almacenes de datos, ha superado con creces nuestra habilidad humana para comprenderla sin herramientas potentes. Como resultado, estos datos recogidos en grandes almacenes de datos se convierten en "tumbas de datos". Consecuentemente, a menudo se toman importantes decisiones basadas, no en los datos llenos de información almacenados, sino en la intuición del responsable, simplemente porque ese responsable no dispone de las herramientas necesarias para extraer el valioso conocimiento contenido en la vasta cantidad de datos. Se han hecho esfuerzos para desarrollar sistemas expertos y tecnologías basadas en conocimiento que, típicamente, dependen de que usuarios o expertos en el campo introduzcan conocimiento manualmente en bases de conocimiento.

Desafortunadamente, sin embargo, la introducción manual de conocimiento es propensa a prejuicios y errores y es extremadamente costosa y laboriosa. La cada vez mayor brecha entre datos e información demanda el desarrollo sistemático de herramientas de minería de datos que puedan convertir tumbas de datos en pepitas de oro de conocimiento.

Mucha gente trata el término minería de datos como un sinónimo de otro popular término, descubrimiento de conocimiento de datos (KDD: *Knowledge Discovery from Data*), mientras que otros entienden la minería de datos únicamente como un paso esencial en el proceso de descubrimiento de conocimiento. El proceso de descubrimiento de conocimiento en una secuencia iterativa que sigue los siguientes pasos:

1 – Limpieza de datos (para quitar ruido y datos inconsistentes).

2 – Integración de datos (se pueden combinar varias fuentes de datos).

3 – Selección de datos (se recuperan datos relevantes para la tarea de análisis de bases de datos).

4 – Transformación de los datos (los datos son transformados y consolidados en la forma apropiada para la minería de datos).

5 – Minería de datos (un proceso esencial en el que se aplican métodos inteligentes para extraer patrones de datos).

6 – Evaluación de los patrones (identificar los patrones realmente interesantes que representan el conocimiento basadas en medidas que sean de interés).

7 – Presentación del conocimiento (se utilizan técnicas de visualización y representación del conocimiento para presentar el conocimiento minado a otros usuarios).

Los pasos del 1 al 4 son diferentes formas de preprocesado de datos, donde los datos son preparados para la minería. El paso de la minería puede interactuar con el usuario o con una base de conocimiento. Los patrones de interés son presentados al usuario y pueden ser almacenados como nuevo conocimiento en la base de conocimiento [8].

2.2 Aplicaciones de la minería de datos

La minería de datos se utiliza ampliamente en diversas áreas. Hay un gran número de sistemas comerciales de minería de datos disponibles y, aun así, todavía hay muchos desafíos en este campo. Las siguientes son solo algunas de las posibles aplicaciones:

2.2.1 Minería de datos aplicada a la medicina

Algunos algoritmos de aprendizaje automático pueden ser aplicados en el campo de la medicina como herramientas de diagnóstico y como herramientas de extracción de conocimiento. Uno de estos clasificadores (PEL-C [9]) es capaz de descubrir síndromes, así como casos clínicos atípicos.

En 2011, el caso de Sorrell v. IMS Health, Inc. [10], juzgado por la Corte Suprema de los Estados Unidos, dictaminó que las farmacéuticas pueden compartir información con compañías externas. Esta práctica fue autorizada bajo la Primera Enmienda a la Constitución de los Estados Unidos, protegiendo la libertad de expresión. Gracias a ello la cantidad de datos médicos disponibles ha crecido enormemente abriendo la puerta a la minería de datos. A raíz de esto, nació una corriente en el campo de la medicina que aboga por una medicina basada en datos. El término designado para nominar esta nueva corriente es “Real-world Evidence”.

2.2.2 Internet de las cosas (IoT)

Los sensores inalámbricos pueden ser utilizados para facilitar la recolección de datos para gran cantidad de aplicaciones tales como monitorización de polución en el aire [11]. Una característica de este tipo de redes es que sensores cercanos entre sí que monitorizan un dato medioambiental concreto tienden a registrar observaciones similares. Este tipo de redundancia debida a la correlación espacial entre las medidas de los sensores inspira técnicas para minería y agregación de datos en una red. Midiendo la correlación espacial entre los datos recolectados por diferentes sensores, una amplia gama de algoritmos especializados puede ser desarrollada para permitir una minería de datos espacial más eficiente.

2.2.3 Educación

Existe un nuevo campo emergente llamado minería de datos para la educación que se refiere al desarrollo de métodos que descubren conocimiento de datos procedentes de entornos educativos. Los objetivos de este campo son: predecir la conducta de aprendizaje futura de un alumno, estudiar los efectos del soporte educacional y progresar en el conocimiento científico acerca del aprendizaje. La minería de datos puede ser utilizada por una institución para tomar decisiones precisas, así como para predecir el resultado de un estudiante. Con estos resultados, la institución puede centrarse en qué enseñar y cómo enseñarlo. Los patrones de aprendizaje de los estudiantes pueden ser capturadas y utilizadas para desarrollar técnicas de enseñanza.

2.2.4 CRM (*Customer Relationship Management*)

CRM, o la gestión de relaciones con los clientes, consiste en adquirir y retener los clientes, mejorar la lealtad de los mismos e implementar estrategias centradas en los clientes. Para poder establecer una relación adecuada con el cliente, un negocio debe recolectar datos y analizarlos. Ahí es donde entra la minería de datos. Utilizando tecnologías de minería de datos, los datos recolectados pueden ser utilizados para análisis. De este modo, en vez de confundir el enfoque, se conseguirán resultados filtrados en la retención de clientes.

2.2.5 Detección de fraude

Miles de millones de dólares se han perdido por culpa de fraudes. Los métodos tradicionales de detección de fraude llevan mucho tiempo y son complejos. La minería de datos puede ayudar encontrando patrones útiles y convirtiendo estos datos en información. Cualquier información que es válida y útil es conocimiento. Un sistema de detección de fraude perfecto debería ser capaz de proteger la información de todos los usuarios. Un método supervisado incluye una colección de registros de ejemplo. Estos registros están clasificados en fraudulentos y no fraudulentos. Se construye un modelo usando estos datos y el algoritmo es desarrollado para identificar si un registro es fraudulento o no.

2.2.6 Banca financiera

Con la banca informatizada en todas partes, una enorme cantidad de datos se genera con cada transacción. La minería de datos puede contribuir a la resolución de problemas en la banca y las finanzas encontrando patrones, causalidades y correlaciones en la información de un negocio o en los precios del mercado que no son inmediatamente evidentes para los administradores porque el volumen de datos con que trabajan es demasiado grande o generado demasiado deprisa para ser cribado por los expertos. Los

administradores pueden utilizar esta información para una mejor segmentación, orientación, adquisición, retención y mantenimiento de un cliente rentable.

2.2.7 Investigación criminal

La criminología es un proceso que trata de identificar las características de un delito. De hecho, el análisis de un crimen consiste en detectar los crímenes, así como su relación con los criminales. La gran cantidad de conjuntos de datos y la gran complejidad de las relaciones entre ellos hace de la criminología el campo apropiado para aplicar técnicas de minería de datos. Informes basados en texto se pueden procesar automáticamente y esta información se puede utilizar para llevar a cabo procesos de búsqueda de casos similares [12].

2.3 Historia de Python

Figura 2.1: Python

Python fue concebido a finales de la década de los 80 y su implementación comenzó en diciembre de 1989 por Guido van Rossum en el Centrum Wiskunde & Informatica (CWI) en los Países Bajos como sucesor del lenguaje ABC (el cual estaba inspirado en SETL), capaz de manejar excepciones y permitiendo interconexión con el sistema operativo Amoeba [13]. Van Rossum es el principal autor de Python y continúa a día de hoy su rol central en el desarrollo del lenguaje como queda reflejado en el título dado por la comunidad de Python, *benevolent dictator for life* [14] (dictador benévolo de por vida).

Python 2.0 fue publicado el 16 de octubre del 2000 y contenía una gran cantidad de grandes mejoras entras las que destaca el colector de basura y soporte para Unicode. Con esta publicación el proceso de desarrollo cambió y empezó a ser más transparente y apoyado por la comunidad.

Python 3.0, una gran publicación incompatible con versiones anteriores, fue lanzada el 3 de diciembre de 2008 tras un largo período de ensayo.

Desde 2003, Python ha estado de manera consistente entre los diez lenguajes de programación más populares de acuerdo al TIOBE Programming Community Index [15]. A fecha de agosto de 2016 era el quinto lenguaje más popular. Fue elegido lenguaje de programación del año en los años 2007 y 2010. Es el tercer lenguaje más popular cuya sintaxis gramatical no está basada predominantemente en C.

Un estudio empírico descubrió que los lenguajes de scripting (como Python o php) son más productivos que los lenguajes tradicionales (como C y Java) a la hora de resolver problemas que involucran la manipulación de cadenas de texto, así como la búsqueda en diccionarios. El uso de memoria es “mejor que en Java y no mucho peor que en C o C++” [16].

Las principales organizaciones que hacen uso de Python son, entre otras, Google, Yahoo!, CERN, NASA, etc. La red social de noticias, Reddit, está escrita en su totalidad en Python.

Bibliotecas como NumPy, SciPy y Matplotlib permiten un uso efectivo de Python en cálculo científico. De cara a este proyecto la librería que más nos interesará es Scikit-learn, una librería de software libre centrada en aprendizaje automático. Incluye varios algoritmos de clasificación, regresión y agrupamiento y está diseñada para ser compatible con NumPy y SciPy [17]. Por último, debemos mencionar Pandas, otra librería de software libre para Python que permite la manipulación y el análisis de datos. En particular, ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales [18].

2.3.1 NumPy

Figura 2.2: NumPy

NumPy [19] es el paquete fundamental para trabajo científico con Python. Contiene, entre otras cosas:

- Un potente array N-dimensional.
- Un conjunto de funciones sofisticadas.
- Herramientas para integrar código de C, C++ y Fortran.
- Capacidades útiles de álgebra lineal, transformada de Fourier y generación de números aleatorios.

Además de los obvios usos científicos, NumPy también puede ser empleado como un eficiente contenedor multidimensional de datos, lo que permite a NumPy integrar rápidamente y sin problemas una amplia variedad de bases de datos.

2.3.2 Pandas

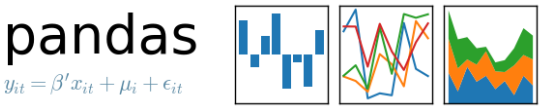

Figura 2.3: Pandas

Pandas [18] es una biblioteca con licencia BSD, de código abierto que ofrece estructuras y herramientas de análisis de datos fáciles de usar y de alto rendimiento para Python.

Python ha sido durante mucho tiempo muy adecuado para limpieza de datos, pero no tanto para análisis y modelado. Pandas ayuda a cubrir este vacío, permitiéndote llevar a cabo todo tu análisis de datos en Python sin necesidad de tener que cambiar a otro lenguaje más específico como R.

Combinado con otras librerías, el entorno para realizar análisis de datos en Python se destaca en el rendimiento, productividad y la posibilidad de colaborar.

2.3.3 Matplotlib

Figura 2.4: Matplotlib

Matplotlib [20] es una biblioteca de Python que permite la representación gráfica en 2D. Trata de hacer fáciles las cosas fáciles y posibles las difíciles. Puede generar gráficos, histogramas, diagramas de barras, etc. con unas cuantas líneas de código.

Provee control sobre todos los elementos del gráfico, desde estilos de fuente a las propiedades de los ejes a través de una interfaz orientada a objetos.

2.3.4 SciPy

SciPy [21] es una colección de algoritmos matemáticos y funciones útiles escritas sobre NumPy. Añade potencia a Python ofreciendo al usuario comandos de alto nivel, así como clases preparadas para la manipulación y visualización de datos. Con Scipy, Python se convierte en un entorno de procesado de datos y creación de sistemas capaz de rivalizar con otro como MATLAB, IDL, Octave, R-Lab y SciLab.

Figura 2.5: SciPy

2.3.5 Scikit-learn

Scikit-learn [17] es un módulo de Python para aprendizaje máquina construido sobre SciPy y distribuido bajo una licencia BSD. El proyecto empezó en 2007 por David Courpeau como proyecto del Google Summer of Code [22], y desde entonces muchos voluntarios han contribuido. Actualmente es mantenido por un equipo de voluntarios.

Figura 2.6: Scikit-learn

Entre sus características destaca:

- Herramientas simples y eficientes para minería de datos y análisis de datos.
- Accesible para todo el mundo y reusable en varios contextos.
- Construido en NumPy, SciPy y matplotlib.
- Código abierto, uso comercional, licencia BSD.

2.4 Metodología CRISP-DM

CRISP-DM [23] (Cross Industry Standard Process for Data Mining) es un modelo de proceso de minería de datos que se divide en seis fases principales:

- Comprensión del negocio

  Esta fase inicial se centra en entender los objetivos del proyecto y los requerimientos desde el punto de vista del negocio para después convertir este conocimiento en la definición de un problema de minería de datos.

- Comprensión de los datos

  La fase de comprensión de los datos empieza con una colección inicial de datos y continúa con actividades que te familiarizan con los datos, te ayudan a identificar problemas y comienzas a entrever posibles ideas.

- Preprocesado de los datos

  Esta fase cubre todas las actividades que tienen como objetivo la construcción del conjunto de final.

- Modelado de datos

  Durante esta fase se seleccionan y aplican varios modelos y sus parámetros son calibrados óptimamente.

- Evaluación

  Llegados a este punto, ya se tiene un modelo que parece tener buena calidad desde el punto de vista del análisis de datos. Sin embargo, es importante revisar el modelo para asegurarse de que cumple los objetivos de negocio impuestos.

- Aplicación

  Dependiendo los requerimientos, la fase de aplicación puede ser desde un reporte de resultados hasta la implementación de un proceso de explotación de información.

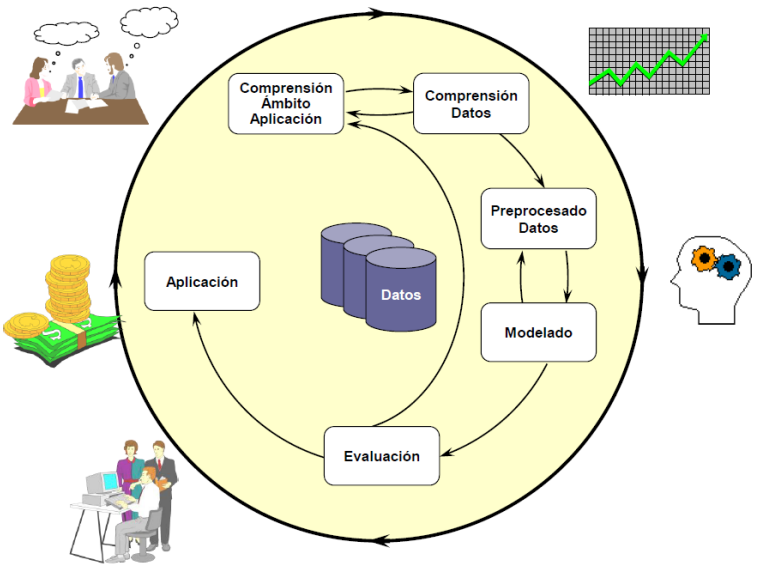

Figura 2.7: Metodología CRISP-DM

3 Diseño e implementación

A la hora de diseñar ADAM se pensó desde un principio en dividirlo en dos partes, una primera que llevase a cabo tareas de limpieza de datos y de búsqueda de datos estadísticos básicos que ayudasen al usuario a entender mejor el conjunto de datos al que se enfrentaba.

3.1 Estructura de ADAM

La estructura básica de ADAM se puede representar de la siguiente manera.

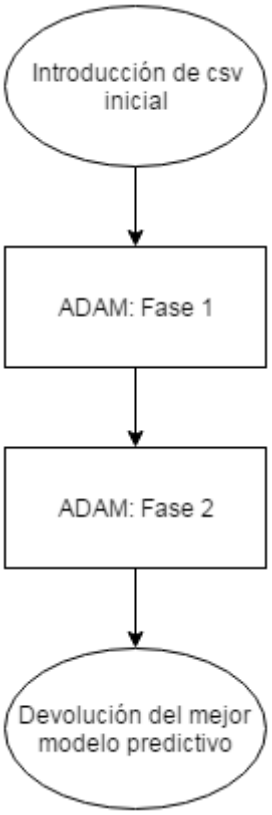

Figura 3.1: Diagrama de flujo: ADAM

El usuario introduce un archivo csv (*comma separated values*) que contiene los datos con los que va a trabajar. Este archivo entrará a la Fase 1 de ADAM en la que se le realizarán una serie de operaciones con la intención de llevar a cabo la tarea de preprocesado. El archivo csv resultante de la fase 1 entrará en la segunda fase en la que se buscará el mejor modelo predictivo de entre los principales y se le devolverá al usuario.

Figura 3.2: Diagrama de flujo: Fase 1

Figura 3.3: Diagrama de flujo: Fase 2

3.2 Términos

En esta sección se explicarán algunos términos y estructuras de datos que se utilizarán durante el proyecto y que mencionaremos en los apartados posteriores:

- DataFrame: Un DataFrame es una estructura tabular perteneciente a la biblioteca pandas. Las columnas de un DataFrame son Series, la estructura básica de pandas (una Serie es una estructura similar a un array que puede contener cualquier objeto). Hay una gran cantidad de funciones definidas para la clase DataFrame que permiten todo tipo de operaciones con los datos, razón por la cuál es la estructura idónea para el desarrollo de ADAM.

- Variable categórica: A lo largo de los próximos apartados haremos referencia a variables categóricas y las diferenciaremos de variables numéricas o continuas. La razón por la que hacemos esta diferenciación es porque no podemos tratar estos dos tipos de variable del mismo modo. Una variable categórica tiene valores que pueden dividirse en un número contable de grupos diferentes. Además, estos datos no tienen ningún orden lógico. Por otro lado, los valores de las variables numéricas o continuas se pueden ordenar y medir. Con esta diferencia en mente, hay que tener en cuenta que un algoritmo de aprendizaje automático no sabe distinguir por sí mismo entre una y otra, lo que provoca que trate como simples números a aquellas variables categóricas denotadas por números. Afortunadamente, hay una buena solución para este problema que consiste en convertir cada una de las categorías en una variable binomial de modo que el algoritmo no cometerá errores.

- Variable sintética: Para mejorar la predicción durante la segunda fase de ADAM, crearemos en la primera una serie de variables sintéticas de las que nos quedaremos con aquellas que tengan una correlación suficientemente alta con la variable objetivo. Se llaman así porque las crearemos a partir del resto de variables realizando operaciones sobre ellas.

4 ADAM: Fase 1. Análisis exploratorio

La Fase 1 de ADAM se divide en una serie de pasos que van encaminados hacia dos objetivos principales: el primero es la creación de un conjunto de datos con el que poder trabajar más adelante y el segundo es aportar al usuario tanta información de su conjunto original como sea posible.

A lo largo de este proceso habrá muchas decisiones con valores por defecto que el usuario podrá modificar y que serán explicadas a media que aparezcan.

4.1 Función data_understanding

```
def data_understanding (csvfile, separator= ";", target_variable = None, id_variable = None, date_variables = [], num_bins = 0, missing_decision = 0)
```

Esta es la función principal de la primera parte de ADAM. Dicha función recibe un archivo csv (*comma separated values*) y devuelve una serie de datos interesantes acerca del mismo, así como un nuevo csv que ha sido limpiado y formateado correctamente. Las tareas que realiza se pueden resumir en:

1 – Extracción de estadísticos básicos de todas las variables (máximo, mínimo, media, mediana…).

2 – Representación de las distribuciones de todas las variables posibles mediante histogramas (solo variables numéricas y categóricas).

3 – Muestra de las correlaciones entre variables.

4 – Generación de variables sintéticas útiles.

5 – Muestra del soporte de todas las posibles combinaciones de variables.

Entradas:

- csvfile: Archivo csv codificado en UTF8 y con los títulos de las columnas en la primera línea.
- target_variable: Nombre de la variable objetivo del csv.
- id_variable: Indica la variable identificadora del conjunto de datos (número o cadena de texto identificadora unívoca de una observación).
- date_variable: Indica la variable o variables de fecha del conjunto de datos.
- separator: Símbolo utilizado en el archivo csv como separador de las columnas.
- num_bins: Número de intervalos que mostrar en el histograma.

- missing_decision: Decide qué hacer con las columnas que contengan valores vacíos.

Salidas:

- df_clean: DataFrame limpio.
- my_vars: Diccionario que contiene todos los estadísticos básicos.

Para llevar a cabo estas tareas, esta función hace uso de otras muchas que comentaremos en adelante. Dichas funciones han sido agrupadas en cuatro módulos.

4.2 Funciones de carga y guardado

```
def data_loading (csvfile, separator = ",")
def data_saving (df, new_csvfile, separator = ",")
```

Estas dos funciones serán necesarias a lo largo del trabajo ya que deberemos leer datos, así como escribirlos.

En el caso de la función “data_loading”, se le debe introducir el nombre de un archivo csv y el separador que utiliza dicho archivo. El separador tiene la “,” como separador por defecto, pero puede ser cambiado por el usuario en cualquier momento.

Entradas:

- csvfile: Nombre del archivo con formato csv.
- separator: Separador utilizado en dicho archivo.

Salidas:

- df: DataFrame cargado.

Por otro lado, la función “data_saving” recibe un DataFrame, el nombre del nuevo archivo csv que queremos crear y el separador que queremos que dicho archivo utilice.

Entradas:

- df: DataFrame de pandas que queremos guardar.
- new_csvfile: Nombre del nuevo archivo csv que se creará.
- separator: Separador que se incluirá en el archivo creado.

4.3 Preparación de los datos

```
data_preparation (df, target_variable = None, date_variables = [])
```

Esta función recibe un DataFrame y le realiza una serie de operaciones para prepararlo para futuros cálculos estadísticos. Inicialmente hace uso de la función "date_parse", que comentaremos más adelante, para convertir las columnas que el usuario introduzca al tipo Datetime de NumPy. Tras esto, busca las variables categóricas contenidas en el DataFrame y se les aplica la función "dummifyColumn" que las convierte en variables binomiales (unos o ceros dependiendo de si cumplen o no la condición de la variable) con las que podremos trabajar más adelante. Esto se hace debido a las limitaciones que tienen las variables categóricas en Python. Por último, esta función toma los valores de fecha generados anteriormente y extrae de ellos tanta información como es posible en nuevas columnas que añade al DataFrame. Estas columnas serán del tipo "Año", "Mes", "Día", "Hora", etc.

Para terminar, se devuelve el DataFrame generado.

Entradas:

- df: DataFrame de entrada que queremos preparar.
- target_variable: Nombre de la variable objetivo del DataFrame.
- date_variables: Nombre de tu variable o variables de fecha.

Salidas:

- df: DataFrame.

4.3.1 Analizador de fechas

```
date_parse (df, date_column, local="us_us")
```

Función utilizada dentro de "data_preparation" que, dada una columna con fechas en cualquier formato (DD/MM/AA, DD/MM/AAAA, etc.) las convierte al formato Datetime [24] de NumPy a partir del cual podremos extraer información extra de cara al futuro análisis.

Entradas:

- df: DataFrame en el que se encuentra tu columna de fecha.
- date_column: Nombre de la columna de fecha que quieres convertir.
- local: Idioma de los datos si necesario (ejemplo: 4/MAY/1977). Inglés elegido por defecto.

Salidas:

- df[col_name]: Columna del DataFrame con la fecha formateada.

4.3.2 Generador de datos a partir de fechas

```
generate_date_data (df)
```

Función que extrae información adicional de todas las columnas de tipo Datetime del DataFrame. Esta información se adjunta al DataFrame en columnas individuales que serán del tipo "nombre original de la columna_day". Se generará información del año, mes, día, hora, minuto y segundo (si existe en el dato original).

Entrada:

- df: DataFrame del que extraer información de fechas.

Salidas:

- df: DataFrame con tanta información recuperada de las columnas de fecha como sea posible.

4.3.3 Binomización de variables

```
def dummifyColumn (df, col, sep = ":")
```

Convierte variables categóricas en una o más binomiales. Es una función necesaria dentro de Python debido a la imposibilidad de trabajar con variables categóricas en la mayoría de los casos. En caso de que la variable solo tenga dos categorías, creará una sola columna con valores True en las posiciones en que la variable origen tome un valor y False en las posiciones del otro valor.

En caso de más de dos categorías posibles creará una columna para cada categoría.

Entradas:

- df: DataFrame en el que se encuentra la columna que quieres convertir en binomial.
- col: Nombre de la columna a convertir en binomial.
- sep: Separador que utilizar en las columnas nuevas.

Figura 4.1: Ejemplo de variable categórica

Figura 4.2: Ejemplo de variable binomizada

Salidas:

- df: DataFrame con tanta información recuperada de las columnas de fecha como sea posible.

```
In [11]: df_train["language"]
Out[11]:
0     English
1     Spanish
2      German
3     English
4     English
5     Spanish
6     English
7     Spanish
8     Spanish
9      German
10    Spanish
11    English
12     German
13    English
14     German
15     German
16    Spanish
17    English
18     German
```

```
In [12]: df_prueba_train["language:Spanish"]
Out[12]:
0     0.0
1     1.0
2     0.0
3     0.0
4     0.0
5     1.0
6     0.0
7     1.0
8     1.0
9     0.0
10    1.0
11    0.0
12    0.0
13    0.0
14    0.0
15    0.0
16    1.0
17    0.0
18    0.0
```

4.4 Cálculo de datos estadísticos básicos

def basicStats (df)

Esta función toma como entrada un DataFrame introducido por el usuario y crea un diccionario de Python con todos los estadísticos básicos que pueda generar para cada una de las variables.

Inicialmente comprobará para cada variable su tipo. En el caso de que sea una variable de tipo cadena de texto (string) o Datetime, la información que generará será el nombre de la variable, el tipo y la cantidad de valores faltantes. Si no es ninguno de estos dos tipos, creará tanto la información ya mencionada como el máximo, mínimo, media, mediana, moda, desviación típica y los percentiles 25, 50, 75 y 100.

El diccionario resultante le será devuelto al usuario.

Entrada:

- df: DataFrame del que generar datos estadísticos.

Salidas:

- my_vars: Diccionario de estadísticos básicos que se devolverá al final.

Clave/Tecla	Tipo	Tamaño	Valor
date	dict	3	{'missing': 0, 'name': 'date', 'type': dtype('<M8[ns]')}
date_day	dict	13	{'max': 31, 'mean': 16.725, 'median': 17.5, 'min': 1, 'missing': 0, 'mode': 0 … dtype: int32, 'name': 'date_day', 'per_100': 31.0, 'per_25': 9.75, 'per_50': 17.…
date_month	dict	13	{'max': 12, 'mean': 5.775, 'median': 6.0, 'min': 1, 'missing': 0, 'mode': 0 1 1 5...8
date_year	dict	13	{'max': 2016, 'mean': 2015.25, 'median': 2015.0, 'min': 2015, 'missing': 0, 'mod… dtype: int32, 'name': 'date_year', 'per_100': 2016.0, 'per_25': 2015.0, 'per_50'…
email	dict	3	{'missing': 7, 'name': 'email', 'type': dtype('O')}
first_name	dict	3	{'missing': 0, 'name': 'first_name', 'type': dtype('O')}
float	dict	13	{'max': 94.0, 'mean': 48.385, 'median': 46.7, 'min': 3.04, 'missing': 0, 'mode':…
gender:Male	dict	13	{'max': 1.0, 'mean': 0.425, 'median': 0.0, 'min': 0.0, 'missing': 0, 'mode': 0 … dtype: float64, 'name': 'gender:Male', 'per_100': 1.0, 'per_25': 0.0, 'per_50': …
id	dict	13	{'max': 40, 'mean': 20.5, 'median': 20.5, 'min': 1, 'missing': 0, 'mode': Series…
integer	dict	13	{'max': 100, 'mean': 53.025, 'median': 60.0, 'min': 1, 'missing': 0, 'mode': 0 … 1 ...8
ip_address	dict	3	{'missing': 0, 'name': 'ip_address', 'type': dtype('O')}
language:English	dict	13	{'max': 1.0, 'mean': 0.4, 'median': 0.0, 'min': 0.0, 'missing': 0, 'mode': 0 … dtype: float64, 'name': 'language:English', 'per_100': 1.0, 'per_25': 0.0, 'per_…
language:German	dict	13	{'max': 1.0, 'mean': 0.325, 'median': 0.0, 'min': 0.0, 'missing': 0, 'mode': 0 … dtype: float64, 'name': 'language:German', 'per_100': 1.0, 'per_25': 0.0, 'per_5…
language:Spanish	dict	13	{'max': 1.0, 'mean': 0.275, 'median': 0.0, 'min': 0.0, 'missing': 0, 'mode': 0 … dtype: float64, 'name': 'language:Spanish', 'per_100': 1.0, 'per_25': 0.0, 'per_…
last_name	dict	3	{'missing': 0, 'name': 'last_name', 'type': dtype('O')}
missing	dict	13	{'max': 89.659999999999997, 'mean': 74.98666666666666, 'median': 78.74, 'min': 5…
target	dict	13	{'max': 2, 'mean': 1.4, 'median': 1.0, 'min': 1, 'missing': 0, 'mode': 0 1 dtype: int64, 'name': 'target', 'per_100': 2.0, 'per_25': 1.0, 'per_50': 1.0, ...

Figura 4.3: Ejemplo de datos estadísticos básicos

4.5 Representación de distribuciones

def barsPlot (df, number_bins = 0)

Dado un DataFrame genera una cuadrícula de N x N tamaño (N será elegido de modo que todas las variables quepan) en el que mostrará todos los histogramas posibles de las distribuciones de las variables en el DataFrame.

Debido a que para ciertas variables no tiene sentido mostrar sus distribuciones (variables del tipo cadena de texto -string- no categóricas como nombres de personas, por ejemplo), se harán una serie de distinciones a la hora de generar los gráficos dependiendo del tipo de la variable.

En primer lugar, si la variable es de tipo Datetime o cadena de texto (string) no categórica, será desestimada y se pasará a la siguiente. Si es de otro tipo y no categórica, se generará un histograma cuyo número de intervalos será decidido por el usuario en la variable de entrada "number_bins". En caso de que el usuario no introduzca el número de intervalos se utilizará el valor por defecto que es la raíz del número de observaciones [25].

$$k = \sqrt{n}$$

k = número de intervalos

n = número de observaciones.

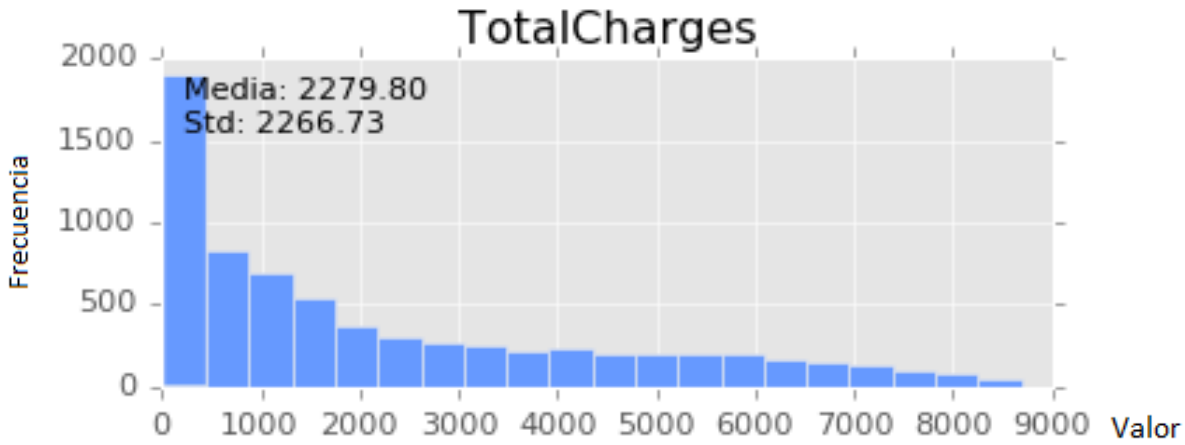

Figura 4.4 Ejemplo de histograma de variable numérica

Por último, para las variables de tipo categórico se creará un intervalo para cada categoría de forma que se pueda observar claramente las diferencias entre ellas.

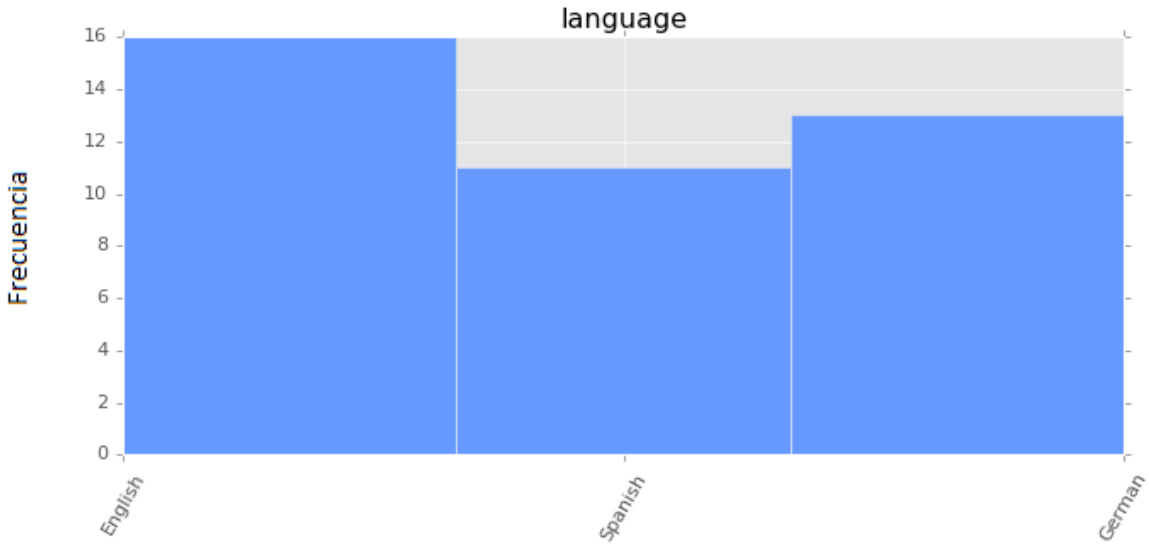

Figura 4.5 Ejemplo de histograma de variable categórica

Entradas:

- df: DataFrame del que representar datos.
- number_bins: Número de intervalos que tendrán los gráficos. En caso de no introducir un valor se utilizará la opción por defecto.

Salidas:

- y_values: Diccionarios de valores de las variables.

4.6 Preparación del DataFrame

```
def prepareDf (df, id_col = None, missing_decision = 0)
```

En este apartado del trabajo, prepararemos el DataFrame para hacer uso de la biblioteca scikit-learn [17] de Python. Lo primero que se hace en este punto es generar una copia del DataFrame a la que haremos cambios con respecto a la original ya que algunos de los datos originales no nos serán de ninguna utilidad estadística en adelante. Tras esto, descartamos la columna identificadora ya que son números o cadenas de texto arbitrarias que no aportan información como variables predictoras. Del mismo modo descartaremos ya las variables que sean cadenas de texto no categóricas y las de tipo Datetime, ya que de estas últimas ya habremos extraído toda la información posible. Por último, aquellas columnas que tengan valores ausentes serán mostradas al usuario junto con la información del porcentaje de valores ausentes y se ofrecerán varias

posibilidades acerca de qué hacer con ellas, desde descartar la columna hasta imputar los valores de diversas formas.

El DataFrame resultante le será devuelto al usuario para continuar el estudio.

Entradas:

- df: DataFrame de entrada a preparar.
- id_col: Nombre de la columna identificadora.
- missing_decision: Decisión acerca de qué hacer con las variables con datos faltantes (valores posibles: 0 – descartar la columna, 1 – imputar con la media, 2 – calcular los valores mediante una regresión).

Salidas:

- df: DataFrame limpio.

```
The column test_column has a 92.50 % of missing values.

Main correlations between predictor variables:
date_day - test_column: 0.93
language:German - language:English: -0.57
language:Spanish - language:English: -0.50
language:Spanish - language:German: -0.43
language:German - integer: -0.34
date_day - language:German: -0.34
date_year - date_month: -0.31

Main correlations with target variable:
target - date_year: 0.35
target - language:Spanish: -0.27
target - date_month: -0.25
target - integer: -0.18
target - language:English: 0.17
target - test_column: 0.12
target - float: -0.11
```

En el ejemplo de la izquierda podemos ver cómo una variable con un 92,5 % de valores ausentes pasa a ser imputada mediante una regresión consiguiendo incluso que su correlación con la variable objetivo sea alta (teniendo en cuenta que estos datos son ficticios).

Figura 4.6 Ejemplo de imputación de valores ausentes.

4.7 Mostrar correlaciones

```
def show_corr (df, target_variable, min_corr_pred = 0.3, min_corr_tg = 0.1, max_number_show = 20)
```

Función que calcula las correlaciones entre todas las variables y muestra las "max_number_show" mayores, siendo el valor por defecto de esta función 20. Las correlaciones son mostradas de mayor a menor valor absoluto y, inicialmente, muestra correlaciones entre las variables predictoras de forma que el usuario pueda comprobar si algunas de sus variables son totalmente dependientes. Tras esto, se mostrarán las correlaciones de las variables predictoras con la variable objetivo del mismo modo que en el caso anterior. Por último, se mostrarán aquellas variables cuya correlación con la variable objetivo sea muy baja o nula. La correlación mínima a mostrar puede ser elegida por el usuario tanto en el caso de correlación entre variables predictoras como en el caso de variable predictora con variable objetivo.

La función devuelve al usuario una lista con todas las variables cuya correlación con la variable objetivo sea considerada como muy baja o nula para que el usuario decida qué hacer con ellas, ya que en el próximo paso se le dará la opción al usuario de descartar estas variables al no ser útiles desde el punto de la predicción.

Entradas:

- df: DataFrame del que mostrar las correlaciones entre variables.
- target_variable: Variable objetivo del conjunto de datos.
- min_corr_pred: Correlación mínima a mostrar entre variables predictoras.
- min_corr_tg: Correlación mínima a mostrar entre las diferentes variables predictoras y la variable objetivo.
- max_number_show: Número máximo de correlaciones a mostrar en cada punto.

Salidas:

- nulls: Lista de variables con correlación baja o nula con la variable objetivo.

```
Main correlations with target variable:
casual - temp: 0.47
casual - atemp: 0.46
casual - humidity: -0.35
casual - workingday:1: -0.32
casual - season:1: -0.24
casual - season:3: 0.19
casual - datetime_year: 0.15
casual - season:2: 0.13
casual - weather:1: 0.12
casual - weather:3: -0.11
```

En las figuras de la izquierda se muestran dos ejemplos de esta función. La figura 4.7 muestra las correlaciones con valor absoluto mayor de 0,1 con la variable objetivo mientras que en la figura 4.8 podemos observar las variables cuya correlación con la variable objetivo es muy baja o nula. Estas última son las variables que descartaremos al no ser útiles

Figura 4.7 Ejemplo de correlación con variable objetivo

```
Null correlations with target variable:
casual - datetime_month: 0.09
casual - windspeed: 0.09
casual - season:4: -0.09
casual - weather:2: -0.06
casual - holiday:1: 0.04
casual - datetime_day: 0.01
casual - weather:4: -0.01
```

Figura 4.8 Ejemplo de correlación nula con variable objetivo

4.7.1 Descarte de variables con correlación muy baja o nula

def dropNullCorrs (df, nulls, target_variable)

En esta función procederemos a descartar aquellas variables que en la función "show_corr" han sido marcadas como de correlación muy baja o nula con la variable objetivo.

Devuelve al usuario el DataFrame de entrada sin las columnas de correlación nula.

Entradas:

- df: DataFrame origen.
- nulls: Lista de variables con correlación nula con la variable objetivo.
- target_variable: Variable objetivo del DataFrame

Salidas:

- df: DataFrame sin las variables con correlación nula.

4.8 Creación de variables sintéticas

```
def syntheticVars (df, target_variable, min_corr = 0.3)
```

Este apartado creará nuevas variables a partir de las variables predictoras originales. Para ello realizaremos operaciones lineales entre las variables predictoras. Dichas operaciones serán de suma, resta, multiplicación y división de columnas. Se creará una nueva variable por cada una de estas operaciones y se comprobará la correlación de cada una de estas nuevas variables con la variable objetivo. En caso de estar por encima de un mínimo preestablecido por el usuario se conservarán, en caso contrario serán descartadas al no proporcionar suficiente información.

Por último, el nuevo DataFrame resultante le será devuelto al usuario.

Entradas:

- df: DataFrame en el que crear las variables.
- target_variable: Variable objetivo del conjunto de datos.
- min_corr: Correlación mínima de las nuevas variables con la variable objetivo para que sean guardadas.

Salidas:

- df: DataFrame con las variables sintéticas útiles.

```
Generating useful synthetic variables.
Main correlations with target variable:
target:2 - date_year: 0.35
target:2 - language:English_date_year_sum: 0.34
target:2 - language:Spanish: -0.27
target:2 - date_month: -0.25
target:2 - integer: -0.18
target:2 - language:English: 0.17
target:2 - missing: 0.12
target:2 - float: -0.11
```

Figura 4.9 Ejemplo de variable sintética útil

4.9 Análisis del soporte de las combinaciones de variables

```
def support (df, support = 0.01, minlen = 1, max_cats = 20)
```

Función que, dado un DataFrame, calcula el soporte de las diferentes posibles combinaciones de valores de las variables. Los calcula y los muestra de menor a mayor número de variables y solo en caso de que el soporte sea mayor al mínimo elegido por el usuario.

Los resultados son mostrados y devueltos al usuario en un DataFrame. Dicho DataFrame contendrá dos columnas, una columna llamada *Pattern* que especifica los valores de las diferentes variables y otra llamada *Support* que muestra el soporte.

El soporte es la proporción de observaciones que contienen una combinación de valores específica.

$$\sup(X) = \frac{|X|}{|D|}$$

Pattern	Support
gender_Female	0.575
gender_Male	0.425
language_English	0.400
language_German	0.325
language_Spanish	0.275
target_1	0.600
target_2	0.400
gender_Female,language_English	0.200
gender_Female,language_German	0.225
gender_Female,language_Spanish	0.150
gender_Female,target_1	0.325
gender_Female,target_2	0.250
gender_Male,language_English	0.200
gender_Male,language_German	0.100
gender_Male,language_Spanish	0.125
gender_Male,target_1	0.275
gender_Male,target_2	0.150
language_English,target_1	0.200
language_English,target_2	0.200
language_German,target_1	0.175
language_German,target_2	0.150
language_Spanish,target_1	0.225
gender_Female,language_English,target_1	0.100
gender_Female,language_English,target_2	0.100
gender_Female,language_German,target_1	0.100
gender_Female,language_German,target_2	0.125
gender_Female,language_Spanish,target_1	0.125
gender_Male,language_English,target_1	0.100
gender_Male,language_English,target_2	0.100
gender_Male,language_Spanish,target_1	0.100

Figura 4.10 Ejemplo de función soporte

En la figura 4.10 se muestra un ejemplo de la salida de la función de análisis del soporte. Por ejemplo, la cantidad de observaciones cuyo género es femenino e idioma alemán es el 0,225 o 22,5 % del total de observaciones, mientras que las observaciones cuyo género es masculino y su idioma es alemán es del 0,100 o 10 %.

Entradas:

- df: DataFrame sobre el que mostrar el soporte de las combinaciones de variables.
- support: Soporte mínimo a mostrar.
- minlen: Longitud mínima.
- max_cats: Número máximo de categorías para las variables categóricas. Necesario porque una variable categórica con demasiadas categorías ralentiza enormemente la ejecución del programa.

Salidas:

- results: DataFrame con las variables sintéticas útiles.

4.10 Estandarización de los datos

def standarizer (df, target_variable)

Función que, dado un DataFrame, estandariza las variables con la intención de poder hacer futuros estudios estadísticos. Devuelve el DataFrame estandarizado a excepción de la variable objetivo que no será modificada.

Entradas:

- df: DataFrame que se estandarizará.
- target_variable: Variable objetivo del conjunto de datos.

Salidas:

- df: DataFrame con las variables sintéticas útiles.

Index	integer	float	date_day
0	4	27.6	31
1	11	78.9	30
2	66	82.7	9
3	60	5.32	29
4	63	32.5	19
5	22	24.4	5

Figura 4.11 Ejemplo estandarización 1

Index	integer	float	date_day
0	-1.78	-0.729	1.65
1	-1.52	1.07	1.54
2	0.47	1.21	-0.894
3	0.253	-1.51	1.42
4	0.361	-0.558	0.263
5	-1.12	-0.843	-1.36

Figura 4.12 Ejemplo estandarización 2

4.11 Funciones auxiliares de uso regular a lo largo del desarrollo

Durante el desarrollo de esta primera fase, ha habido una serie de funciones de uso recurrente que, pese a no haber sido mencionadas antes, han sido necesarias:

4.11.1 Función para el cambio de tipos

```
def change_types (df, percentage = 0.1, max_cats = 20)
```

Función que, dado un DataFrame, comprueba si el número de categorías en cada variable cumple la condición impuesta para que esa variable sea considerada como categórica. Dicha condición consiste en que el número total de categorías de la variable sea inferior a un porcentaje del número de datos que será definido por el usuario. El valor por defecto de este porcentaje será del 0,1 (10 %). Por otro lado, se impondrá una segunda condición que vendrá dada por el número máximo de categorías que el usuario quiere que tenga una variable. Una vez más, existe un valor por defecto que será de 20 categorías.

Entradas:

- df: DataFrame de entrada sobre el que vamos a trabajar.
- percentage: Porcentaje máximo de categorías sobre el total de los datos.
- max_cats: Número máximo de categorías.

Salidas:

- df: DataFrame con las variables categóricas convertidas en binomiales.

4.11.2 Ordenar DataFrame

```
def order_dataframe (df,target_variable)
```

Esta función reordena un DataFrame colocando en último lugar la variable objetivo. Es una pequeña función principalmente útil de cara a poder identificar la variable objetivo fácilmente en otras funciones y así poder tratar de diferente manera a variables predictoras y objetivo

Entradas:

- df: DataFrame que queremos ordenar.
- target_variable: Variable objetivo. Será colocada en última posición en el DataFrame.

Salidas:

- df: DataFrame con las variables ordenadas.

4.11.3 Variables categóricas

```
def categorical_vars (df, percentage = 0.1, max_cats = 20)
```

En esta función tomamos un DataFrame y devolvemos al usuario una lista de las variables categóricas del DataFrame. Las condiciones aplicadas para considerar o no una variable como categórica son las mismas que las ya aplicadas previamente en la función “change_types”.

Entradas:

- df: DataFrame del que recuperar las variables categóricas.
- percentage: Porcentaje máximo de categorías sobre el total de los datos.
- max_cats: Número máximo de categorías.

5 ADAM: Fase 2. Entrenamiento de datos

Durante esta segunda fase de ADAM procederemos a centrarnos en la parte de predicción o clasificación. Está desarrollado con la intención de que cualquier DataFrame pase primero por la Fase 1 para así conseguir un DataFrame limpio que será el que utilizaremos en esta segunda parte.

Una vez más, como en el caso anterior, hay una serie de decisiones que el usuario podrá cambiar, pero siempre tendrán un valor por defecto.

5.1 Función data_training

```
def data_training(csvfile, target_variable, separator = ';' , id_variable = None)
```

Función principal de la Fase 2 de ADAM. Recibe un DataFrame que ha sido limpiado por la Fase 1 de ADAM y sobre el que se aplicarán una serie de funciones con el objetivo de conseguir el mejor modelo posible de predicción.

Los pasos que seguirá esta función son los siguientes:

1 – Reordenación del conjunto de datos.

2 – Binomización de variables categóricas que aún no lo hayan sido.

3 – Estandarización.

4 – Muestra de las variables con mayor fuerza predictora.

5 – Búsqueda del mejor modelo de predicción.

Entradas:

- csvfile: Fichero que contiene el DataFrame limpio.
- separator: Símbolo utilizado en el fichero csv para separar las columnas.
- target_variable: Define la variable objetivo.
- id_variable: Indica la variable identificadora, si la hay, del conjunto de datos.

Salidas:

- model: Modelo que mejor predice con los datos dados.

Al igual que en el caso de la Fase 1, para realizar todas estas tareas se hace uso de una serie de funciones que han sido agrupadas en cuatro módulos diferentes: depuration, preparation, statistics y visualization.

5.2 Binomización de variables categóricas restantes

def dummifyPredictors (df, target_variable)

Esta función hace uso de la ya definida función "categorical_vars" para comprobar si falta alguna variable por convertir en binomial y, en caso de ser así, las convierte en binomiales mediante la función "dummifyColumn".

Finalmente devuelve al usuario el DataFrame con todas las variables convertidas en categóricas.

Entradas:

- df: DataFrame que comprobar.
- target_variable: Variable objetivo.

Salidas:

- df: DataFrame con las variables predictoras binomizadas.

5.3 Cálculo de las variables con mayor fuerza predictiva

def var_scoring (df, target_variable, id_variable = None)

Función que entrena un modelo de scikit-learn llamado Ridge [26] con el objetivo de mostrar los coeficientes de regresión al usuario dándole así una idea previa acerca de qué variables tienen una mayor o menor fuerza predictiva. Una vez calculados, hace uso de la función "print_coeffs" que detallaremos más adelante y devuelve al usuario el dataframe que contiene los coeficientes.

Entradas:

- df: DataFrame del que se desea conocer información.
- target_variable: Variable objetivo sobre la que se entrenará el regresor.
- id_variable: Variable identificador. Se descartará al no ser útil como variable predictora.

Salidas:

- df_coeffs: Lista de los coeficientes de regresión.

5.3.1 Muestra de los coeficientes previamente calculados

def print_coeffs (df_coeffs, max_show = 20)

Recibe un DataFrame que contiene una fila por cada variable predictora con su correspondiente coeficiente de regresión. Dicho DataFrame ya ha sido ordenado de mayor a menor valor absoluto de los coeficientes, por lo que esta función solo debe mostrar los "max_show" primeros coeficientes, siendo 20 el número por defecto a mostrar.

```
Main predictor variables for Churn:Yes

TotalCharges: -0.0972
tenure: -0.0491
InternetService:Fiber optic: 0.0395
InternetService:DSL: -0.0368
PaymentMethod:Electronic check: 0.0263
PaperlessBilling:Yes: 0.0246
SeniorCitizen:1: 0.0234
Contract:Month-to-month: 0.0195
Contract:One year: -0.0182
MonthlyCharges: 0.0163
Dependents:Yes: -0.0161
PaymentMethod:Mailed check: -0.0130
PaymentMethod:Bank transfer (automatic): -0.0126
StreamingTV:Yes: 0.0125
MultipleLines:No: -0.0119
OnlineSecurity:No: 0.0114
TechSupport:No: 0.0114
StreamingMovies:Yes: 0.0114
MultipleLines:Yes: 0.0097
StreamingTV:No: -0.0081
```

Figura 5.1 Ejemplo de coeficientes de regresión

Entradas:

- df_coeffs: DataFrame que contiene las variables predictivas junto al valor de sus respectivos coeficientes.
- max_show: Número máximo de coeficientes a mostrar. Por defecto será 20.

5.4 Búsqueda del mejor modelo predictivo

def best_model (df, classification, target_variable)

Función que, dado un DataFrame y la decisión acerca de si utilizar modelos de clasificación o de regresión, separa los datos en predictores y objetivo y utiliza funcionalidades de la librería scikit-learn entre las que destaca "GridSearchCV" [27], una función que entrena el modelo que le indiques haciendo uso de validación cruzada y cruzando, también, los parámetros del modelo que el usuario le indique. Es una herramienta muy potente gracias a la cual el trabajo de prueba de algoritmos por parte del usuario se ve reducida en gran medida.

Los principales algoritmos son entrenados y se obtiene la puntuación de cada uno de ellos para los parámetros óptimos. Por último, el algoritmo que mejor puntuación obtenga será elegido y devuelto al usuario para que pueda pasar a predecir haciendo uso de este.

En el caso de la clasificación los algoritmos que se entrenarán son:

- kNN Classifier: Una observación es asignada a la clase más común entre sus k vecinos más cercanos.
- Logistic Regression: La regresión logística mide la relación entre la variable categórica dependiente y una o más variables predictoras estimando probabilidades mediante una función logística.
- SVC (Support Vector Classifier): Un modelo SVM (Support Vector Machine o Máquina de Soporte Vectorial) representa las observaciones como puntos en el espacio mapeados de forma que las categorías estén separadas tanto como sea posible. Las nuevas observaciones serán entonces mapeadas en ese espacio.
- Random Forest Classifier: Funciona construyendo una gran cantidad de árboles de decisión. En el caso de la clasificación, la observación a predecir será decidida por la moda de los árboles mientras que en regresión será la media de las predicciones de los árboles.

Mientras que en el caso de que se aplique regresión:

- kNN Regressor: La salida es la media de los valores de sus k vecinos más cercanos.
- Linear Regression
- SVR (Support Vector Regressor): Explicado arriba
- Random Forest Regressor: Explicado arriba.
- Gradient Boosting Regressor: Produce un modelo de predicción a partir de un conjunto de modelos. Construye dicho modelo por etapas al igual que otros algoritmos de impulso y lo generaliza permitiendo la optimización de una función de coste arbitraria.

Entradas:

- df: DataFrame sobre el que buscar el modelo óptimo.
- classification: Variable binaria. En caso de True se aplicará clasificación; en caso de False se aplicará regresión.
- target_variable: Variable objetivo del conjunto de datos.

Salidas:

- model: Modelo que mejor predice con los datos dados.

5.4.1 Decisión acerca de si utilizar modelos de clasificación o de regresión

Justo antes de buscar el mejor modelo, se comprueba si la variable objetivo es categórica o continua. En caso de ser detectada como categórica, se entrenarán modelos de clasificación mientras que, si es detectada como continua, se entrenarán modelos de regresión.

En cualquier caso, estos son datos que pueden ser modificados por el usuario en caso de que este así lo estime.

```
if( target_variable in prep.categorical_vars(df_norm) ):

    classification = True
elif( df_norm[target_variable].dtype in cont_types ):

    classification = False
```

Figura 5.2 Código decisión clasificación o regresión

6 Pruebas

Para comprobar el buen funcionamiento de ADAM, vamos a hacer pruebas con dos conjuntos de datos diferentes, uno de ellos contiene datos de una empresa de telecomunicación mientras que el segundo, que pertenece a la plataforma Kaggle, contiene datos de alquiler de bicis en Washington.

En primer lugar, vamos a echar un vistazo a los datos antes de entrar en ADAM.

```
In [14]: df_telco.columns.tolist()
    ...:
Out[14]:
['customerID',
 'gender',
 'SeniorCitizen',
 'Partner',
 'Dependents',
 'tenure',
 'PhoneService',
 'MultipleLines',
 'InternetService',
 'OnlineSecurity',
 'OnlineBackup',
 'DeviceProtection',
 'TechSupport',
 'StreamingTV',
 'StreamingMovies',
 'Contract',
 'PaperlessBilling',
 'PaymentMethod',
 'MonthlyCharges',
 'TotalCharges',
 'Churn']
```

Figura 6.1 Columnas conjunto de datos empresa telecomunicaciones

En este conjunto de datos, nuestro objetivo será predecir la variable “Churn”. Es una variable binaria que indica si un cliente de esta compañía se ha ido o no, por lo tanto, es un caso de clasificación.

En este segundo conjunto, nuestro objetivo será predecir la variable “casual”. Es una variable numérica que indica el número de bicis que se alquilan en una determinada hora, por lo tanto, es un caso de regresión.

```
In [15]: df_bicis.columns.tolist()
Out[15]:
['datetime',
 'season',
 'holiday',
 'workingday',
 'weather',
 'temp',
 'atemp',
 'humidity',
 'windspeed',
 'casual']
```

Figura 6.2 Columnas conjunto de datos alquiler de bicis

En la primera fase de ADAM, los DataFrames pasan por la primera etapa de limpieza que, como ya hemos explicado más arriba, consiste en extraer información de las variables de fecha, convertir las variables categóricas en binomiales y ordenar el DataFrame resultante.

```
Out[32]:
['customerID',
 'tenure',
 'MonthlyCharges',
 'TotalCharges',
 'gender:Male',
 'SeniorCitizen:1',
 'Partner:Yes',
 'Dependents:Yes',
 'PhoneService:Yes',
 'MultipleLines:No',
 'MultipleLines:No phone service',
 'MultipleLines:Yes',
 'InternetService:DSL',
 'InternetService:Fiber optic',
 'InternetService:No',
 'OnlineSecurity:No',
 'OnlineSecurity:No internet service',
 'OnlineSecurity:Yes',
 'OnlineBackup:No',
 'OnlineBackup:No internet service',
 'OnlineBackup:Yes',
 'DeviceProtection:No',
 'DeviceProtection:No internet service',
 'DeviceProtection:Yes',
 'TechSupport:No',
 'TechSupport:No internet service',
 'TechSupport:Yes',
 'StreamingTV:No',
 'StreamingTV:No internet service',
 'StreamingTV:Yes',
 'StreamingMovies:No',
 'StreamingMovies:No internet service',
 'StreamingMovies:Yes',
 'Contract:Month-to-month',
 'Contract:One year',
 'Contract:Two year',
 'PaperlessBilling:Yes',
 'PaymentMethod:Bank transfer (automatic)',
 'PaymentMethod:Credit card (automatic)',
 'PaymentMethod:Electronic check',
 'PaymentMethod:Mailed check',
 'Churn:Yes']
```

Figura 6.3 Columnas formateadas conjunto telecomunicaciones

En este conjunto no había ninguna variable de fecha sin embargo había muchas categóricas que han sido descompuestas en variables binomiales con nombres del tipo “nombre_original:valor”.

En este otro caso, se ha extraído información de la variable temporal “datetime” así como se han descompuesto en binarias las variables categóricas.

```
Out[36]:
['datetime',
 'temp',
 'atemp',
 'humidity',
 'windspeed',
 'season:1',
 'season:2',
 'season:3',
 'season:4',
 'holiday:1',
 'workingday:1',
 'weather:1',
 'weather:2',
 'weather:3',
 'weather:4',
 'datetime_day',
 'datetime_month',
 'datetime_year',
 'casual']
```

Figura 6.4 Columnas formateadas conjunto bicis

Tras este primer formateado de los datos, pasamos a calcular estadísticos básicos, así como a representar las distribuciones de las variables.

Clave/Tecla	Tipo	Tamaño	Valor
max	float64	1	41.0
mean	float	1	20.230859819952173
median	float	1	20.5
min	float64	1	0.81999999999999995
missing	int32	1	0
mode	Series	(1,)	0 14.76 dtype: float64
name	str	1	temp
per_100	float64	1	41.0
per_25	float64	1	13.94
per_50	float64	1	20.5
per_75	float64	1	26.239999999999998
std	float	1	7.791589843987506
type	dtype	1	dtype('float64')

Figura 6.5 Ejemplo de estadísticos básicos

Este es un ejemplo de los estadísticos básicos que ADAM recupera de cada una de las variables.

A continuación, se muestra un ejemplo de las representaciones de los valores de las variables que genera ADAM.

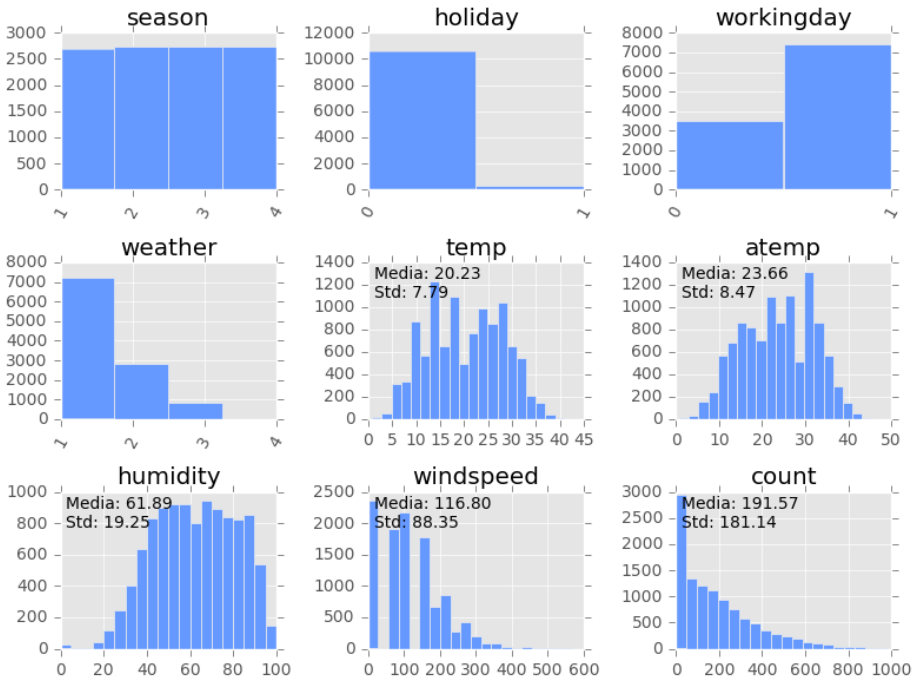

Figura 6.6 Ejemplo de histograma

Tras esto, pasamos a la segunda etapa de preparación de los datos en la que descartaremos las variables de tipo string no categóricas, así como la variable identificadora al no aportar información útil desde el punto de vista de la predicción. Una vez hecho esto pasamos a generar variables sintéticas y mostrar las correlaciones entre nuestras variables.

```
In [62]: df_bicis = stat.syntheticVars(df_bicis, target_variable_bicis)

Generating useful synthetic variables.
```

Generamos tantas variables sintéticas como sean necesarias. El requerimiento que deben cumplir es tener una correlación mayor a un mínimo dado.

Tras ello, pasamos a mostrar las correlaciones de todas nuestras variables con la variable objetivo.

```
Main correlations with target variable:
casual - atemp_humidity_sub: 0.50
casual - temp_humidity_sub: 0.49
casual - temp_workingday:1_sub: 0.49
casual - atemp_workingday:1_sub: 0.48
casual - temp_datetime_year_sum: 0.47
casual - temp_season:3_sub: 0.47
casual - temp_atemp_mult: 0.47
casual - temp_weather:1_sum: 0.47
casual - temp_weather:3_sub: 0.47
casual - temp_season:2_sum: 0.47
casual - temp_season:1_sum: 0.47
casual - atemp_datetime_year_sum: 0.47
casual - temp_datetime_year_mult: 0.47
casual - temp: 0.47
casual - temp_datetime_year_div: 0.47
casual - atemp_weather:1_sum: 0.47
casual - atemp_season:3_sub: 0.47
casual - temp_atemp_sum: 0.47
casual - temp_season:1_sub: 0.47
casual - atemp_weather:3_sub: 0.46
```

Figura 6.7 Ejemplo de correlaciones

Hecho esto, ya tenemos limpios y correctamente formateados nuestros conjuntos de datos y podemos pasar a la segunda fase de ADAM.

Inicialmente se nos muestran las variables con mayor poder predictivo.

```
Main predictor variables for Churn:Yes

TotalCharges: -0.1226
MonthlyCharges_TotalCharges_div: 0.0839
MonthlyCharges_Contract:Month-to-month_mult: -0.0652
MonthlyCharges: 0.0576
InternetService:Fiber optic_Contract:Month-to-month_mult: 0.0549
TechSupport:No_Contract:Month-to-month_mult: 0.0342
tenure: 0.0271
OnlineBackup:No_Contract:Month-to-month_mult: 0.0244
StreamingTV:No: -0.0224
OnlineSecurity:No_TechSupport:No_mult: 0.0219
OnlineSecurity:No_Contract:Month-to-month_mult: 0.0208
StreamingMovies:No: -0.0181
OnlineSecurity:No_PaperlessBilling:Yes_mult: 0.0168
PaperlessBilling:Yes_PaymentMethod:Electronic check_mult: 0.0151
TechSupport:No_PaperlessBilling:Yes_mult: 0.0145
InternetService:Fiber optic_OnlineBackup:No_mult: -0.0144
MonthlyCharges_PaymentMethod:Electronic check_mult: -0.0142
OnlineBackup:No_PaymentMethod:Electronic check_mult: 0.0120
InternetService:Fiber optic_OnlineSecurity:No_mult: 0.0100
Contract:Month-to-month_PaperlessBilling:Yes_mult: 0.0099
```

Figura 6.8 Ejemplo de variables con mayor poder predictivo

Después, pasamos a decidir cuál es el modelo que mejor predice con nuestros datos.

```
Training models for Churn:Yes

Best parameters found for kNN Classifier: {'n_neighbors': 83}
The score achieved for it is: 0.7894

Best parameters found for Logistic Regression: {'C': 0.01}
The score achieved for it is: 0.8040

Best parameters found for SVC: {'kernel': 'linear', 'C': 0.01}
The score achieved for it is: 0.7969

Best parameters found for Random Forest Classifier: {'n_estimators': 500}
The score achieved for it is: 0.7841
```

Figura 6.9 Ejemplo de elección del mejor modelo

Como podemos ver el que consigue mejor puntuación en este caso es la Regresión Logística.

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=0, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

Figura 6.10 Mejor modelo elegido

7 Conclusions and future lines of work

7.1 Conclusions

This Final Degree Project presents ADAM, a framework able to clean, format and analyze a given data set helping data scientist with their work. It is divided in two main stages, the first one takes a raw data set and performs all kind of data cleaning techniques to return the user a new set of data they can use for further analysis. Another one of the main tasks that Stage 1 performs is the preliminary analysis of the data set. Finally, Stage 2 will take the cleaned set of data that Stage 1 returns and it will test several algorithms in the search for the best model.

The programming language used to develop ADAM was Python. There are several reasons behind this choice, but the main ones are the existence of some libraries that offer tools that are really useful for us. Some of these libraries are NumPy, Pandas or Scikit-learn.

During Stage 1, the user will provide a csv (comma separated values) file and the method will return several interesting statistics as well as a new csv file ready for Stage 2. The main tasks Stage 1 performs can be summed up as:

1 – Calculation of basic statistics regarding every variable (maximum, minimum, mean, mode, standard deviation, median…).

2 – Plot of the distribution of every possible variable (continuous and categorical variables).

3 – Calculation and display of the main correlations between variables.

4 – Creation of useful synthetic variables.

5 – Support for every possible combination of variables.

Stage 2 is given a clean DataFrame (the output of Stage 1) and it will perform a number of methods on the data in order to get the best predictive model.

The steps that follow this function are:

1 – Reorder the data set.

2 – Transform every categorical variable into binomials.

3 – Standardization of the data.

4 – Display of the variables with the highest predictive strength.

5 – Search for the best predictive model.

In order to perform all these tasks, many more have been developed, and they have been grouped up in four modules.

All the objectives were accomplished with very few time delays. Once finished, ADAM was tested over several real data from the platform Kaggle [28] with relatively good results taking into account this is just the first iteration of ADAM.

7.2 Future lines of work

As future lines of work, most of the functions can be improved further by making them more customizable and allowing the user many more possibilities. Some examples of these improvements are allowing the user to choose the cost function to minimize, giving the possibility for the user to choose how they want every variable to be processed, improving the speed of both stages, parallelizing processes, maybe even implementing it all in Apache Spark for cluster computing.

Another challenge to tackle would be the development of another Stage of ADAM regarding unsupervised learning, allowing clustering and some of the main algorithms.

Finally, there is the need of a graphical user interface for the program so that it becomes easier for the general public to use.

Anexo A: Presupuesto y plan de trabajo

En este anexo se va a analizar tanto el tiempo empleado para elaborar el proyecto, así como el presupuesto destinado al mismo. Los recursos humanos se contabilizarán a partir de las horas trabajadas por el alumno asignando un valor económico a cada hora.

A.1 Recursos humanos y coste de material

Como material empleado para el proyecto, se contará únicamente el coste de un ordenador como el utilizado para el desarrollo de la aplicación. Al haber sido programado en Python utilizando software libre, no hay ningún otro costo adicional.

Tabla A.1: Coste del material

Ordenador	600 €
Monitor	136.86 €
Total	736.86 €

El total de horas invertidas por el alumno en el proyecto es de un total aproximado de 400 horas. Se añade una remuneración de 15 € por hora. El total de horas invertidas por el supervisor del proyecto es de 90 horas, con un salario de 35 € por hora.

Tabla A.2: Horas trabajadas

Recursos Humanos	Coste/Hora	Dedicación	Coste Total
Alumno del proyecto	15 €/hora	400 horas	6000 €
Supervisor del proyecto	35 €/hora	90 horas	3150 €
Total	-	-	9150 €

El coste total del proyecto ascendería a 9886.86 €. Finalmente se muestra un diagrama de Gannt con la planificación del proyecto, en el cual se tiene en cuenta desde el estudio previo hasta la escritura de la memoria.

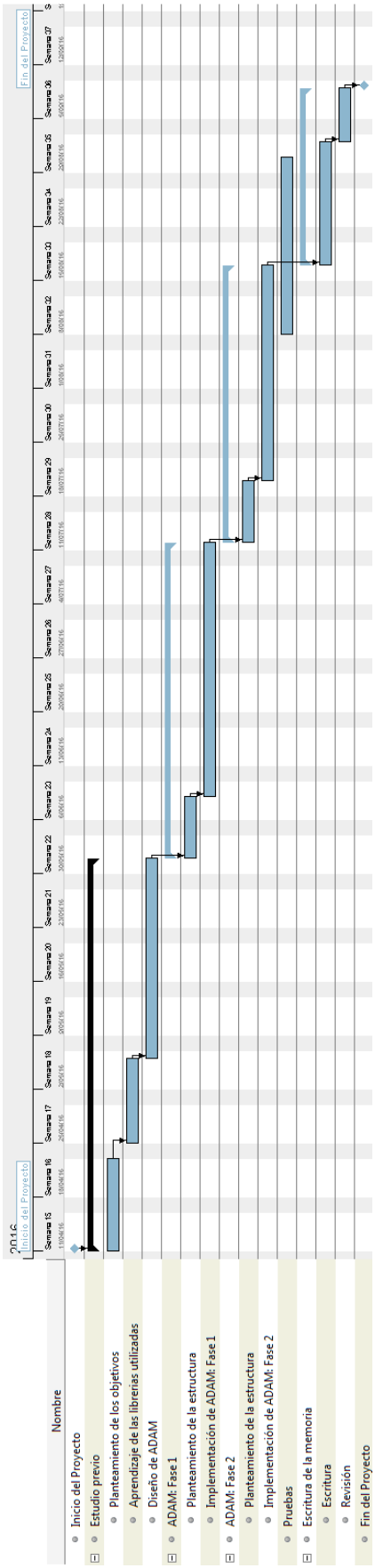

Figura A.1: Diagrama de Gannt

Anexo B: Extended Summary

B.1 Introduction

Big Data is a concept that refers to the storage of massive amounts of data and the procedures used to find patterns within the data.

The discipline dedicated to Big Data falls within the field of information and communications technology. This discipline takes responsibility for all activities related to systems handling large data sets. The most common difficulties associated with the management of these data are the collection and storage, search, sharing, analysis and visualization.

The upper limit of processing has grown over the years. Thus, the limits set in 2008 were around the order of petabytes or zettabytes [1]. These limitations also affect Internet search engines, finance systems and business computing. Data sets grow in volume due to the massive data collection from wireless sensors and mobile devices, the steady growth of application logs, cameras, microphones… Technological capacity per capita worldwide to store data doubles, approximately, every forty months since the eighties. It is estimated that in 2012, 2.5 quintillion bytes were created each day [2].

In order to deal with all these problems a new field was born: Data Science. Data Science is an interdisciplinary field that involves processes and systems to extract knowledge or a better understanding of large volumes of data in its different forms (both structured and unstructured) and formats.

It is a new paradigm on which researchers rely on systems and processes very different to those used in the past as models, equations, algorithms, as well as evaluation and interpretation of results.

It is estimated that 60 % of the time [4] of a data scientist is spent cleaning and organizing data. For this reason, it would be interesting the creation of an application capable of performing part of this work automatically so that data scientists can devote a bigger part of their time to improving their algorithms.

B.2 Objective

The main objective pursued is the creation of a framework capable of analyzing, cleaning and extracting as much information as possible from any given data set. Said framework will be called ADAM (Automated Discovery and Analysis Machine). It will be a python script divided in two main stages:

Stage 1 will return the user a series of useful statistics regarding his data set as well as perform operations of data cleaning for a later usage of the resulting dataframe.

- Basic statistics: A python dictionary will be created containing as much information of every variable as possible. This information will contain name of the variable, type, amount of missing values, maximum value, minimum value, mean, median, mode, standard deviation and the 25th, 50th, 75th and 100th percentiles. Lastly, a plot of every possible histogram will be displayed.

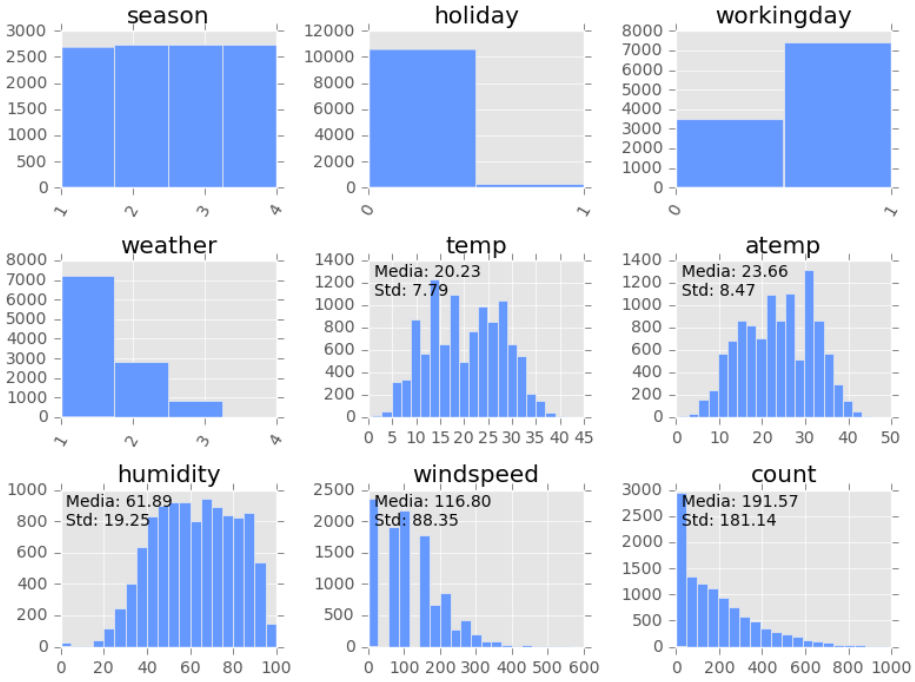

Figure B.1: Example of a Histogram

- Correlations: Main correlations between variables will be shown. First off, correlations between predictor variables will be displayed. After that two list of correlations with the target variable will be presented, the first one with the highest correlations (both positive and negative) of every predictor variable with the target variable while the second list will contain every null correlation with the target. Finally, these null correlation variables will be dropped since they will not be useful when predicting.

- Variables value support: Computes the support for every possible combination of values being the support the proportion of items in the data set containing that specific combination.

$$\sup(X) = \frac{|X|}{|D|}$$

- Creation and analysis of synthetic variables: Finally, some synthetic variables will be generated. A synthetic variable is a variable created by performing some kind of operation to two or more of the original variables. This framework will perform operations of addition, subtraction and multiplication of variables and will check whether the correlation of these new variables with the target variable is good enough to keep them or not.

Stage 2 will take a dataset clean and it will return a trained model ready to be used.

- Scoring of predictor variables: A SGD Regressor (Stochastic Gradient Descent) will be performed to the data in order to show the user the variables with the highest predictive value.
- Search for the optimal prediction model: First off, it will be checked whether the target variable is categorical or not in order to perform either regression or classification algorithms. Once that information is known, several algorithms will be tested and the one performing best will be chosen and returned to the user.

B.3 State of Art

B.3.1 Data Science

Many people treat the term data mining as a synonym of another popular term, Knowledge Discovery from Data (KDD), while others understand data mining just as an essential step in the process of knowledge discovery. The process of knowledge discovery is an iterative sequence that follows these steps:

1 – Data cleaning (in order to get rid of noise and inconsistent data).

2 – Data integration (several data sources can be combined into a dataset).

3 – Data selection (relevant data regarding the task of analysis are recovered from the database).

4 – Data transformation (data are transformed and consolidated into the appropriated form for data mining).

5 – Data mining (an essential process where intelligent methods are applied to extract patterns from the data).

6 – Pattern evaluation (identify those patterns that are actually interesting and that represent knowledge).

7 – Knowledge presentation (visualization and representation techniques used to present knowledge to other users).

Steps 1 through 4 are different forms of data preprocessing, where data are prepared for mining. The data mining step can interact with the user or a knowledge base. The interesting patterns are presented to the user and can be stored as new knowledge in the knowledge base [8].

B.3.3 Applications of data mining.

Data mining is used widely in a number of fields. There is a large number of commercial systems of data mining available and yet there are still many challenges in this field. The following are just some of the possible applications:

B.3.3.1 Data mining in medicine

Some machine learning algorithms can be applied in the medicine field as diagnosis and knowledge extraction tools. Thanks to the case of Sorrell v. IMS Health, Inc. [10] in 2011, the is a huge amount of medical data available, opening the door to data mining.

B.3.3.2 Internet of things (IoT)

Wireless sensors can be used for collecting and monitoring data for a wide variety of applications such as air pollution monitoring [11]. These kind of networks produce a redundancy in data due to the spatial correlation between them that allows the usage of a class of algorithms specialized yielding a more efficient data mining.

B.3.3.3 Education

In the field of education, the objectives of data mining are to predict the future learning behavior of a student and to study the effects of educational support and progress towards a more scientific knowledge about learning. The learning patterns of students can be stored and used to develop new teaching techniques.

B.3.3.4 Customer relationship management (CRM)

CRM is to acquire and retain customers, improve their loyalty and implement strategies focused on them. In order to do so, a business has to collect and analyze data.

B.3.3.5 Fraud detection

Traditional fraud detection systems are too complex. Data mining can help finding patterns and transforming raw data into information. A perfect fraud detection system should be able protect everyone's information while detecting fraud correctly.

B.3.3.6 Financial banking

The computerization of banking has brought a vast amount of data that has to be analyzed. Data mining can help solving problems by finding patterns, causalities, correlations in the information of a business and predicting market prices.

B.3.3.7 Criminal investigation

Criminology is the process of identifying the characteristics of a crime. There is a huge amount of data sets and the relationships between them are complex. That makes criminology the appropriated field for data mining [12].

B.3.1 History of Python

Figure B.2: Python

Python was envisaged in the second half of the eighties and its implementation did not start until December 1989 by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands. It was designed as a successor to the ABC language able to handle exceptions and allowing interfacing with the Amoeba operating system [13].

Since 2003, Python has consistently been amongst the top ten most popular programming languages according to TIOBE Programming Community Index [15].

In an empirical study carried out it was shown that scripting languages such us Python were more productive than traditional languages when manipulating strings and searching in dictionaries. The memory usage was "better than Java and not much worse than C or C++" [16].

They main organizations to make use of Python are, amongst others, Google, Yahoo!, CERN, NASA, etc. The social news networking site, Reddit, is written entirely in Python.

Libraries such as NumPy, SciPy and Matplotlib allow an effective usage of Python in scientific calculus. Regarding this project, the most interesting library is Scikit-learn, an open software library focused on machine learning. It includes algorithms regarding regression, classification and clustering and it's designed to be compatible with NumPy and SciPy [21]. Lastly, we must mention Pandas, another open software library for Python that allows manipulation and analysis of data. In particular, it offers data structures and operations to handle numeric tables and time series [18].

B.3.1.1 NumPy

Figure B.3: NumPy

NumPy is the main module for scientific work in Python. It contains, amongst other things:

- A powerful multi-dimensional array.
- Sophisticated functions.
- Tools to integrate C, C++ and Fortran code.
- Useful linear algebra, Fourier transform and random number capabilities.

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional data container. This allows NumPy to integrate easily and speedily with a number of databases [19].

B.3.1.2 Pandas

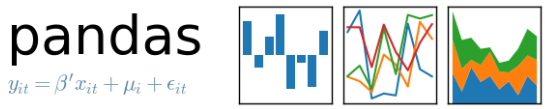

Figure B.4: Pandas

Pandas is a BSD license, open source library that offers structures and data analysis tools with a high performance and easy to use for Python.

Python has been for a long time the perfect data munging tool, but not so much for analysis and modeling. Pandas helps fill that gap allowing you to carry out your entire data analysis in Python without having to change switch to any other language more specific like R.

Combined with other libraries, the environment to carry out data analysis in Python excels in performance, productivity and the ability to collaborate [18].

B.3.1.3 Scikit-learn

Scikit-learn is a Python package for machine learning built over SciPy and distributed under a BSD license. The project started in 2007 by David Courpeau as a Google Summer of Code project [22], and since then it has been developed by a series of volunteers. To this day, it is maintained and developed by volunteers.

Figure B.5: Scikit-learn

Amongst its most powerful tools are:

- Simple and efficient data mining and data analysis tools.
- Accessible to everybody and reusable in various contexts.
- Built on NumPy, SciPy and matplotlib.
- Open source, commercially usable, BSD licensed [17].

B.3.4 CRISP-DM Methodology

CRISP-DM [23] (Cross Industry Standard Process for Data Mining) is a data mining process that is divided in six main phases:

- Business understanding

  This is the initial phase. It focuses on understanding the objective of the project and the requirements from the business point of view in order to transform those requirements into a data science problem.

- Data understanding

  The data understanding phase starts with an initial dataset and continues with activities that allow the user to get to know the data.

- Data preparation

  This phase covers all the activities that aim towards building the final dataset.
- Data modeling

  During this phase several models will be chosen and applied and their parameters calibrated.
- Data evaluation

  At this point, we already have a model with a seemingly good quality from an analysis point of view. Now, we have to make sure it fulfills the objectives set.
- Deployment

  Depending on the requirements, the deployment phase can be from generating a report to the implementation of a data mining process.

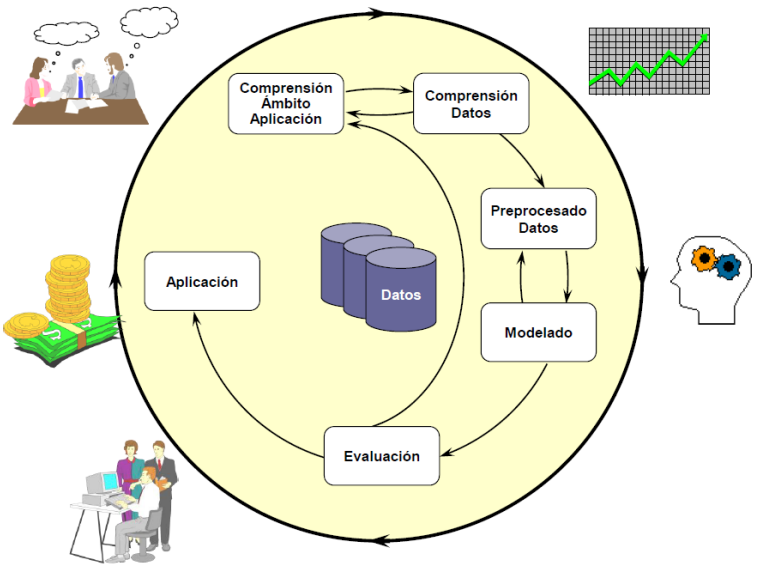

Figure B.6: CRISP-DM Methodology

B.4 ADAM: Stage 1. Data Analysis

ADAM: Stage 1 is divided in a series of steps towards two main objectives: the creation of a dataset ready to be worked with later on and the second is to provide the user with as much information as possible regarding his original set of data.

Through this process there will be several decisions regarding default values that the user will be able to modify.

```
def data_understanding (csvfile, separator= ";", target_variable = None, id_variable = None, date_variables = [], num_bins = 0, missing_decision = 0)
```

This is the main function of ADAM: Stage 1. Said function receives a csv (comma separated values) file which first row must be dedicated to the names of the variables. A number of statistics will be calculated and displayed to the user and the new csv file, clean and well formatted, will be returned. The main tasks carried out can be summed up as:

1 – Extraction of basic statistics of every variable (maximum, minimum, mean, mode, standard deviation, median…).

2 – Graphical representation of the distribution of every possible variable using histograms (numeric and categorical variables only).

3 – Display of correlations between variables.

4 – Creation of useful synthetic variables.

5 – Support of every variable combination about a threshold.

Entradas:

- csvfile: csv file coded using UTF8 and with the column's names in the first line.
- target_variable: Name of the target variable.
- id_variable: Denotes the identification variable of the data set.
- date_variable: Denotes the variable or variables containing date information.
- separator: Symbol used in the csv file as columns separator.
- num_bins: Number of bins for the histograms to show.
- missing_decision: Decides what to do with the columns containing missing values.

In order to go through these tasks, this function makes use of many others that have been grouped up in four groups: depuration, preparation, statistics and visualization.

B.5 ADAM: Stage 2. Data training

During ADAM: Stage 2 our focus will be to develop a software able to perform prediction over a data set. Since the project is developed with the intention for someone to use it as a whole, it is expected for any DataFrame to go through Stage 1 first in order to get the best DataFrame for this stage to work as intended. Anyway, in case that is not the case, there will be some functions that will try to clean as much as possible a raw input.

Once more, as in the previous step, there will be decisions that the user will be able to modify but that will always have a default value.

```
def data_training(csvfile, target_variable, separator = ';' , id_variable = None)
```

Main function of ADAM: Stage 2. It receives a DataFrame which is assumed to have been cleaned by ADAM: Stage 1 and on which a series of functions will be applied with the intent of achieving the best predicting model.

The basic steps followed by this function are:

1 – Reorder of the data set.

2 – Change categorical variables into binomial ones.

3 – Standardization.

4 – Display of the variables with the highest predictive strength.

5 – Search of the best predicting model.

Entradas:

- csvfile: File containing the clean DataFrame.
- separator: Symbol used in the csv file as columns separator.
- target_variable: Denotes the name of the target variable.
- id_variable: Denotes the identification variable of the data set.

As in Stage 1, in order to perform all these tasks, we make use of several functions grouped up in four modules: depuration, preparation, statistics and visualization.

B.6 Conclusion

This Final Degree Project presents ADAM, a framework able to clean, format and analyze a given data set helping data scientist with their work. It's divided in two main stages, the first one takes a raw data set and performs all kind of data cleaning techniques to return the user a new set of data they can use for further analysis. Another one of the main tasks Stage 1 performs is the preliminary analysis of the data set. Finally, Stage 2 will take the clean set of data Stage 1 returns and it will test several algorithms in the search for the best model.

The programming language used to develop ADAM was Python. There are several reasons behind this choice, but the main ones are the existence of some libraries that offer tools that are really useful for us. Some of these libraries are NumPy, Pandas or Scikit-learn.

During Stage 1, the user will introduce a csv (comma separated values) file and the method will return several interesting statistics as well as a new csv file ready for Stage 2. The main tasks Stage 1 performs can be summed up as:

1 – Calculation of basic statistics regarding every variable (maximum, minimum, mean, mode, standard deviation, median…).

2 – Plot of the distribution of every possible variable (continuous and categorical variables).

3 – Calculation and display of the main correlations between variables.

4 – Creation of useful synthetic variables.

5 – Support for every possible combination of variables.

Stage 2 is given a clean DataFrame (the output of Stage 1) and it will perform a number of methods on the date in order to get the best predictive model.

The steps that follow this function are:

1 – Reorder the data set.

2 – Transform every categorical variable into binomials.

3 – Standardization of the data.

4 – Display of the variables with the highest predictive strength.

5 – Search for the best predictive model.

In order to perform all these tasks, many more have been developed, and they have been grouped up in the following four modules: depuration, preparation, statistics and visualization.

References

[1] Horowitz, M. 2008. Visualizing Big Data: Bar Charts for Words. [ONLINE] Available at: https://www.wired.com/2008/06/pb-visualizing/##ixzz0llT2DN5j. [Accessed 9 September 2016].

[2] IBM. 2016. IBM - What is Big Data? [ONLINE] Available at: http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html. [Accessed 9 September 2016].

[3] Liu, A. 2016. Data Science and Data Scientist. [ONLINE] Available at: http://www.researchmethods.org/DataScienceDataScientists.pdf. [Accessed 9 September 2016].

[4] Gil Press. 2016. *Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says*. [ONLINE] Available at: http://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/. [Accessed 7 September 2016].

[5] Laerd. 2016. Pearson Product-Moment Coefficient. [ONLINE] Available at:https://statistics.laerd.com/statistical-guides/pearson-correlation-coefficient-statistical-guide.php. [Accessed 9 September 2016].

[6] Páez, A., Le Gallo, J. (2012). *Using Synthetic Variables in Instrumental Variable Estimation of Spatial Series Models.* Université de Franche-Comté, Besançon, France. School of Geography and Earth Sciences, Ontario, Canada.

[7] European Commission. 2016. Protection of personal data. [ONLINE] Available at: http://ec.europa.eu/justice/data-protection/. [Accessed 9 September 2016].

[8] Han, Jiawei, and Micheline Kamber. *Data Mining: Concepts and Techniques*. San Francisco: Morgan Kaufmann, 2001.

[9] Gagliardi, F. (2011). *Instance-based classifiers applied to medical databases: Diagnosis and knowledge extraction*. University of Rome "La Sapienza", Rome, Italy

[10] Oyez. 2009. *Sorrell v. IMS Health Inc*. [ONLINE] Available at: https://www.oyez.org/cases/2010/10-779. [Accessed 7 September 2016].

[11] Ma, Y., Richards, M., Ghanem, M., Guo, Y. and Hassard, J. (2008). *Air Pollution Monitoring and Mining Based on Sensor Grid in London*. Imperial College London, London, United Kingdom.

[12] Rajkumar P. 2014. *14 useful applications of data mining*. [ONLINE] Available at: http://bigdata-madesimple.com/14-useful-applications-of-data-mining/. [Accessed 7 September 2016].

[13] General Python FAQ. 2016. *General Python FAQ*. [ONLINE] Available at: https://docs.python.org/3/faq/general.html#why-was-python-created-in-the-first-place. [Accessed 7 September 2016].

[14] The Python Wiki. 2014. *BDFL - Python Wiki*. [ONLINE] Available at: https://wiki.python.org/moin/BDFL. [Accessed 7 September 2016].

[15] TIOBE. 2016. *TIOBE Index for August 2016*. [ONLINE] Available at: http://www.tiobe.com/tiobe-index/. [Accessed 7 September 2016].

[16] Pretzel, L. (2000). *An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl*. Universität Karlsruhe, Karlsruhe, Germany.

[17] scikit-learn. 2016. *scikit-learn: machine learning in python*. [ONLINE] Available at: http://scikit-learn.org/stable/. [Accessed 7 September 2016].

[18] Python Data Analysis Library. 2016. *Python Data Analysis Library*. [ONLINE] Available at: http://pandas.pydata.org/index.html. [Accessed 7 September 2016].

[19] NumPy. 2016. *NumPy*. [ONLINE] Available at: http://www.numpy.org/. [Accessed 7 September 2016].

[20] Matplotlib. 2016. matplotlib. [ONLINE] Available at: http://matplotlib.org/. [Accessed 9 September 2016].

[21] SciPy. 2016. Introduction - SciPy. [ONLINE] Available at: http://docs.scipy.org/doc/scipy/reference/tutorial/general.html. [Accessed 9 September 2016].

[22] Google. 2016. Google Summer of Code. [ONLINE] Available at: https://developers.google.com/open-source/gsoc/. [Accessed 9 September 2016].

[23] Chapman, P. 2016. The CRISP-DM user guide. [ONLINE] Available at: http://lyle.smu.edu/~mhd/8331f03/crisp.pdf. [Accessed 9 September 2016].

[24] SciPy.org. 2016. *Datetimes and Timedeltas*. [ONLINE] Available at: http://docs.scipy.org/doc/numpy/reference/arrays.datetime.html. [Accessed 7 September 2016].

[25] Colin Cameron, A. 2009. *EXCEL Univariate: Histogram*. [ONLINE] Available at: http://cameron.econ.ucdavis.edu/excel/ex11histogram.html. [Accessed 7 September 2016].

[26] sklearn. 2016. Ridge. [ONLINE] Available at: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html. [Accessed 9 September 2016].

[27] sklearn. 2016. GridSearchCV. [ONLINE] Available at: http://scikit-learn.org/stable/modules/generated/sklearn.grid_search.GridSearchCV.html. [Accessed 9 September 2016].

[28] Kaggle. 2016. Kaggle. [ONLINE] Available at: https://www.kaggle.com/. [Accessed 9 September 2016].